Global and Partitioned Multiprocessor Fixed Priority Scheduling with Deferred Pre-emption

ROBERT I. DAVIS, University of York
ALAN BURNS, University of York
JOSE MARINHO, CISTER/INESC-TEC, ISEP
VINCENT NELIS, CISTER/INESC-TEC, ISEP
STEFAN M. PETTERS, CISTER/INESC-TEC, ISEP
MARKO BERTOGNA, University of Modena

This paper introduces schedulability analysis for global fixed priority scheduling with deferred pre-emption (gFPDS) for homogeneous multiprocessor systems. gFPDS is a superset of global fixed priority pre-emptive scheduling (gFPPS) and global fixed priority non-pre-emptive scheduling (gFPNS). We show how schedulability can be improved using gFPDS via appropriate choice of priority assignment and final non-pre-emptive region lengths, and provide algorithms which optimize schedulability in this way. Via an experimental evaluation we compare the performance of multiprocessor scheduling using global approaches: gFPDS, gFPPS, and gFPNS, and also partitioned approaches employing FPDS, FPPS, and FPNS on each processor.

Categories and Subject Descriptors: **C.3 [SPECIAL PURPOSE AND APPLICATION-BASED SYSTEMS] Real-time and embedded systems**

General Terms: Algorithms, Performance, Theory, Verification

Additional Key Words and Phrases: Deferred Pre-emption, Limited Pre-emption, Global Scheduling, Partitioned Scheduling, Fixed Priority , Real-Time; Multiprocessor; Multicore;

ACM Reference Format:

Robert Davis, Alan Burns, Jose Marinho, Vincent Nelis, Stefan Petters, Marko Bertogna, 2014 Global and Partitioned Multiprocessor Fixed Priority Scheduling with Deferred Pre-emption *ACM Trans. Embedd. Comput. Syst.* X, X, Article XX (XXXX 2014), 25 pages.
DOI:http://dx.doi.org/10.1145/0000000.0000000

EXTENSIONS

This paper both extends and revises the research presented in "*Global Fixed Priority Scheduling with Deferred Pre-emption"* [Davis et al. 2013]. The simple deadline and response time based schedulability tests for global fixed priority scheduling with deferred pre-emption (gFPDS) given in that paper factored in the effects of push-through blocking due to the final non-pre-emptive region of the previous job of the same task; however, the more sophisticated versions of those tests that limit 'carry-in' interference omitted to do so. As a result, the limited carry-in tests could give

This work was partially funded by the UK EPSRC Tempo project (EP/G055548/1), the UK EPSRC MCC project (EP/K011626/1), and by Portuguese National Funds through FCT (Portuguese Foundation for Science and Technology), and by ERDF (European Regional Development Fund) through COMPETE (Operational Programme 'Thematic Factors of Competitiveness'), within the RePoMuC project, (FCOMP-01-0124-FEDER-015050).

Author's addresses: R.I.Davis and A. Burns, University of York, UK;. J. Marinho, V. Nelis, S.M. Petters CISTER/INESC-TEC, ISEP, Porto, Portugal; M. Bertogna, University of Modena, Italy.

optimistic results. In this paper, we address this issue, correcting the formulation of those tests by including an extra term accounting for push-through blocking. (We prove limits on the number of 'carry-in' jobs in Lemma 1 and Lemma 2 and hence bound the interference from them). The repercussions of this revision on the sustainability of the tests are also addressed (Theorems 2, 3 and 4). Further, we investigate partitioned scheduling using deferred pre-emption, (pFPDS) with an experimental evaluation of the performance of pFPDS with respect to partitioned fully-pre-emptive (pFPPS) and non-pre-emptive scheduling (pFPNS).

1. INTRODUCTION

A common misconception with regard to fixed priority scheduling of sporadic tasks is that fully pre-emptive scheduling is more effective in terms of schedulability than non-pre-emptive scheduling. The two are however incomparable; there are tasksets that are schedulable under fixed priority non-pre-emptive scheduling that are not schedulable under fixed priority pre-emptive scheduling and vice-versa. This is the case for uniprocessor scheduling [Davis and Bertogna 2012] and also the case for global multiprocessor scheduling [Guan et al. 2011], which is the main focus of this paper.

While the blocking effect, due to long non-pre-emptive regions of low priority tasks, degrades schedulability for single processor systems that have a wide range of task execution times and periods (as illustrated by Figure 7 in [Davis and Bertogna 2012]), [Guan et al. 2011] showed that the same is not necessarily true for multiprocessor systems. With m processors rather than one, long non-pre-emptive regions can be accommodated without necessarily compromising the schedulability of higher priority tasks. However, this advantage only extends so far; with m processors then m long non-pre-emptive regions are enough to significantly compromise schedulability. In this context, limited non-pre-emptive execution has the advantage of reducing the number of pre-emptions, and potentially improving the worst-case response time of tasks, while also keeping blocking effects on higher priority tasks within tolerable limits.

With partitioned multiprocessor scheduling, this effect is also apparent, as our evaluations show. In this case, task allocation can place tasks with broadly similar parameters (e.g. execution times and deadlines) on the same processor, thus enabling the use of relatively long non-pre-emptive regions, while other tasks with much shorter deadlines are allocated to a different processor.

In the literature, the term *fixed priority scheduling with deferred pre-emption* has been used to refer to a variety of different techniques by which pre-emptions may be deferred for some interval of time after a higher priority task becomes ready. These are described in a survey by [Buttazzo et al. 2013] and briefly discussed in Section 2. In this paper, we assume a simple form of fixed priority scheduling with deferred pre-emption where each task has a single non-pre-emptive region at the end of its execution. If this region is of the minimum possible length for all tasks, then we have fully pre-emptive scheduling, whereas if it constitutes all of the task's execution time then we have non-pre-emptive scheduling.

In this paper, we introduce sufficient schedulability tests for global fixed priority scheduling with deferred pre-emption (gFPDS). gFPDS can be viewed as a superset of both global fixed priority pre-emptive scheduling (gFPPS) and global fixed priority non-pre-emptive scheduling (gFPNS) and strictly dominates both. With gFPDS, there are two key parameters that affect schedulability: the priority assigned to each task, and the length of each task's final non-pre-emptive region (FNR). The FNR length affects both the schedulability of the task itself, and the schedulability of tasks with higher priorities. This is a trade-off as increasing the FNR length can improve

schedulability for the task itself by reducing the number of times it can be pre-empted, but potentially increases the blocking effect on higher priority tasks which may reduce their schedulability.

[Davis and Bertogna 2012] introduced an optimal algorithm for fixed priority scheduling with deferred pre-emption on a single processor. This algorithm finds a schedulable priority assignment and set of FNR lengths whenever such a schedulable combination exists. In this paper we also build upon this work, extending it to the multiprocessor case. For a given priority ordering, we show how to find an assignment of FNR lengths that results in a system that is deemed schedulable under gFPDS according to our sufficient schedulability tests, whenever such an assignment of FNR lengths exists. We also show that the *Final Non-pre-emptive Region and Priority Assignment* (*FNR-PA*) algorithm from [Davis and Bertogna 2012] is *not optimal* in the multiprocessor case, but nevertheless can be used as a heuristic for determining both priority ordering and final non-pre-emptive region lengths.

Finally, we also apply the optimal single processor FPDS techniques of [Davis and Bertogna 2012] directly to the multiprocessor case via partitioned fixed priority scheduling with deferred pre-emption (pFPDS). This enables us to make an experimental evaluation of the performance of both global and partitioned fixed priority scheduling with deferred pre-emption, with respect to their fully pre-emptive and non-pre-emptive counterparts.

2. BACKGROUND RESEARCH

2.1 Deferred pre-emption

Two different models of fixed priority scheduling with deferred pre-emption have been developed in the literature.

In the *fixed* model, introduced by [Burns 1994], the location of each non-pre-emptive region is statically determined prior to execution. Pre-emption is only permitted at pre-defined locations in the code of each task, referred to as *pre-emption points*. This method is also referred to as *co-operative scheduling*, as tasks co-operate, providing re-scheduling / pre-emption points to improve schedulability.

In the *floating* model [Baruah 2005; Yao et al. 2009; Marinho et al. 2013] an upper bound is given on the length of the longest non-pre-emptive region of each task. However, the location of each non-pre-emptive region is not known a priori and may vary at run-time, for example under the control of the operating system.

For uniprocessor systems, exact schedulability analysis for the fixed model was derived by [Bril et al. 2009]. Subsequently, pre-emption costs and cache related pre-emption delays (CRPD) were integrated into analysis of the fixed model, considering both fixed [Bertogna et al. 2010] and variable [Bertogna et al. 2011a] pre-emption costs. [Bertogna et al. 2011b] derived a method for computing the optimal FNR length of each task in order to maximize schedulability assuming a given priority assignment. [Davis and Bertogna 2012] introduced an optimal algorithm that is able to find a schedulable combination of priority assignment and FNR lengths whenever such a schedulable combination exists.

2.2 Global fixed priority scheduling

[Baker 2003] developed a strategy that underpins an extensive thread of subsequent research into schedulability tests for gFPPS [Baruah and Fisher 2008; Bertogna et al. 2005; Bertogna and Cirenei 2007; Bertogna et al. 2009; Fisher and Baruah 2006; Guan et al. 2009], and gFPNS [Guan et al. 2011]. (For a comprehensive survey of multiprocessor real-time scheduling, the reader is referred to [Davis and Burns

2011c]). Baker's work was subsequently built upon by [Bertogna et al. 2005; 2009]. They developed sufficient schedulability tests for gFPPS based on bounding the maximum workload in a given interval. [Bertogna and Cirinei 2007] adapted this approach to iteratively compute an upper bound on the response time of each task, using the upper bound response times of other tasks to limit the amount of interference considered. [Guan et al. 2009] extended this approach using ideas from [Baruah 2007] to limit the amount of carry-in interference.

[Davis and Burns 2009; 2011a] showed that priority assignment is fundamental to the effectiveness of gFPPS. They proved that Audsley's Optimal Priority Assignment (OPA) algorithm [Audsley 1991; 2011] is applicable to some of the sufficient tests developed for gFPPS, including the deadline-based test of [Bertogna et al. 2009], but not to others such as the later response time tests [Bertogna and Cirinei 2007; Guan et al. 2009].

[Guan et al. 2011] provided schedulability analysis for gFPNS based on the approach of [Baker 2003], and the techniques introduced by [Bertogna et al. 2005].

gFPDS is broadly similar to the dynamic algorithm FPZL [Davis and Burns 2011b; Davis and Kato 2012]. FPZL resembles gFPPS until a job reaches a state of zero laxity i.e. when its remaining execution time is equal to the elapsed time to its deadline. FPZL gives such a job the highest priority, and hence makes it non-pre-emptable. The length of time each job spends executing in this zero-laxity state is determined dynamically by FPZL. With FPZL, RTOS support for this dynamic behaviour is required, whereas with gFPDS the transition to non-pre-emptive execution may be controlled either by the RTOS, or via API calls suitably located within the code of each task.

[Block et al. 2007] introduced the idea of *link-based* scheduling0, which uses a lazy pre-emption mechanism with the aim of avoiding issues of repeated blocking which can occur when tasks execute non-pre-emptive regions under global multiprocessor scheduling. While the original context for this work was resource locking protocols, the approach also applies to general non-pre-emptive regions as discussed by [Brandenburg 2011] in section 3.3.3 of his thesis. Depending on the task parameters and non-pre-emptive region lengths, the lazy pre-emption mechanism of link-based scheduling may improve or diminish schedulability compared to global fixed priority scheduling with eager pre-emption. Further discussion of link-based scheduling, and an example of such incomparability are given in the appendix.

3. SYSTEM MODEL, TERMINOLOGY AND NOTATION

In this paper, we are mainly interested in global fixed priority scheduling of an application on a homogeneous multiprocessor system with m identical processors. The application or taskset is assumed to consist of a static set of n tasks ($\tau_1 ... \tau_n$), with each task τ_i assigned a unique priority i, from 1 to n (where n is the lowest priority). We assume a discrete time model, where all task parameters are positive integers (e.g. processor clock cycles). We use the notation $hp(i)$ (and $lp(i)$) to mean the set of tasks with priorities higher than (lower than) i.

Tasks are assumed to comply with the *sporadic* task model. In this model, each task gives rise to a potentially unbounded sequence of jobs. Each job may arrive at any time once a minimum inter-arrival time has elapsed since the arrival of the previous job of the same task.

Each task τ_i is characterised by its relative *deadline* D_i, *worst-case execution time* C_i ($C_i \le D_i$), and minimum inter-arrival time or *period* T_i. It is assumed that all tasks have constrained deadlines ($D_i \le T_i$). The *utilisation* U_i of each task is given by C_i / T_i. Under gFPDS, each task is assumed to have a final non-pre-emptive region of length F_i in the range $[1, C_i]$ (here, the minimum value is 1 rather than 0 as a task

can only be pre-empted at discrete times corresponding to processor clock cycles). Finding an appropriate FNR length for each task is assumed to be part of the scheduling problem.

The *worst-case response time* R_i of a task is the longest possible time from the release of the task until it completes execution. Thus task τ_i is schedulable if and only if $R_i \leq D_i$ and a taskset is schedulable if and only if $\forall i \quad R_i \leq D_i$. We use R_i^{UB} to indicate an upper bound on the worst-case response time of task τ_i .

Under gFPDS, at any given time, the m ready tasks with the highest priorities are selected for execution. Final non-pre-emptive regions are assumed to be implemented by manipulating task priorities, thus a task executing its FNR has the highest priority and will not be pre-empted.

The tasks are assumed to be independent and so cannot be blocked from executing by another task, other than due to contention for the processors. Further, it is assumed that once a job starts to execute it will not voluntarily suspend itself.

Job parallelism is not permitted; hence, at any given time, each job may execute on at most one processor. As a result of pre-emption and subsequent resumption, a job may migrate from one processor to another. The costs of pre-emption, migration, and the run-time operation of the scheduler are assumed to be either negligible, or subsumed within the worst-case execution time of each task. (Pre-emption costs are an issue we aim to address in future work).

A taskset is said to be *schedulable* with respect to some scheduling algorithm, if all valid sequences of jobs that may be generated by the taskset can be scheduled by the algorithm without any missed deadlines.

A priority assignment policy P is said to be *optimal* with respect to a schedulability test for some type of fixed priority scheduling algorithm (e.g. gFPPS, gFPNS, or gFPDS) if there are no tasksets that are deemed schedulable, according to the test, under the scheduling algorithm using any other priority ordering policy, that are not also deemed schedulable with the priority assignment determined by policy P.

4. SCHEDULABILITY ANALYSIS FOR gFPDS

In this section, we introduce sufficient schedulability tests for global fixed priority scheduling with deferred pre-emption (gFPDS).

On a uniprocessor, under fixed priority scheduling with deferred pre-emption, a higher priority task can only be blocked by a single job of a lower priority task that starts executing non-pre-emptively prior to the release of a job of the higher priority task. With global scheduling, the multiprocessor case is however significantly different. This is illustrated by Figure 1 below, for the case of 4 processors. Here, a job of the task of interest τ_k (priority 2) is released at time $t = 1$, along with a job of the higher priority task τ_1. τ_k is unable to execute initially due to blocking from three jobs of lower priority tasks (τ_3, τ_4, and τ_5) that have entered their FNRs (shown in dark grey in Figure 1). At time $t = 4$, τ_k begins executing. At $t = 7$, three further jobs of lower priority tasks (τ_6, τ_7, and τ_5 again) enter their FNRs. At $t = 8$, τ_k is pre-empted by a second job of τ_1 and misses its deadline at $t = 12$.

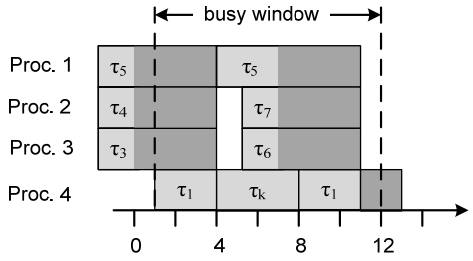

Figure 1: Blocking effect due to FNRs of lower priority jobs.

This example serves to illustrate the following:

- Multiple lower priority tasks may contribute interference in the *problem window* [Baker 2003] of the job of interest. (The problem window is some interval of time at the end of which a deadline is missed, for example from the release time to the deadline of some job of task τ_k). Further, the number of lower priority tasks that may contribute is not limited to m as it is in the non-pre-emptive case [Guan et al. 2011].
- Multiple jobs of the same lower priority task may contribute interference, due to the fact that the task of interest does not occupy all of the processors when it executes; unlike in the uniprocessor case.
- If there were multiple non-pre-emptive regions within each lower priority task, then each of these regions could potentially contribute interference. (This is easy to see by assuming that all of the execution of task τ_5 on processor 1 belongs to one job rather than two).

We note that, in the example in Figure 1 the lazy pre-emption mechanism of link-based scheduling [Block et al. 2007; Brandenburg 2011] would prevent the second job of task τ_1 from pre-empting task τ_k enabling the latter to meet its deadline. In general; however, schedulability with lazy and eager pre-emption mechanisms is incomparable due to trade-offs between blocking effects as shown in the appendix. In the remainder of this paper, we consider only scheduling with the eager pre-emption mechanism.

While no worst-case scenario is currently known, we can obtain an upper bound on the interference from the non-pre-emptive execution of lower priority tasks, by modelling this non-pre-emptive execution as a set of virtual tasks executing at the highest priority. Thus for each lower priority task $\tau_i \in lp(k)$, we assume a virtual task τ_{iv} with the following parameters: $C_{iv} = F_i - 1$, $T_{iv} = T_i$, $D_{iv} = D_i$, $R_{iv}^{UB} = R_i^{UB}$ and the highest priority. (We note that $C_{iv} = F_i - 1$ as the task must have actually entered its FNR in order to be non-pre-emptable).

We note the following points regarding schedulability of task τ_k under gFPDS:

1. Once task τ_k enters its FNR it will execute to completion. Hence with gFPDS if we can show that the task is guaranteed to execute for $C_k^* = C_k - (F_k - 1)$ within an effective deadline of $D_k^* = D_k - (F_k - 1)$, then it is guaranteed to execute for C_k by its deadline D_k.
2. Virtual tasks representing the FNRs of lower priority tasks can effectively be released at any point during the interval in which the corresponding lower priority task may execute. Thus limitations on the number of tasks with *carry-in jobs*[1] [Davis and Burns 2011a] do not apply to virtual tasks.

[1]A carry-in job is defined as a job that is released strictly prior to the start of the interval of interest, and causes interference within that interval.

3. The FNR of the previous job of the task τ_k may cause push-through blocking[2] [Davis et al. 2007]. Push-through blocking can delay execution of one or more higher priority tasks beyond the release of the job of task τ_k that we are interested in, potentially increasing its response time.

In the following sub-sections we give schedulability tests for tasks executing under gFPDS.

4.1 Deadline Analysis for gFPDS

We now extend and adapt the deadline-based, schedulability test of Bertogna et al. (Theorem 8 in [Bertogna et al. 2009]) to gFPDS. Under gFPPS, if task τ_k is schedulable in an interval of length L, with an execution time of C, then an upper bound on the interference over the interval due to a higher priority task τ_i with a carry-in job is given by the following equation Bertogna et al. 2009]. (Note the D superscript denotes *Deadline Analysis*).

$$I_i^D(L,C) = \min(W_i^D(L), L - C + 1) \tag{1}$$

where $W_i^D(L)$ is an upper bound on the workload of task τ_i in an interval of length L (see Figure 2), given by:

$$W_i^D(L) = N_i^D(L)C_i + \min(C_i, L + D_i - C_i - N_i^D(L)T_i) \tag{2}$$

and $N_i^D(L)$ is the maximum number of jobs of task τ_i that contribute all of their execution time in the interval:

$$N_i^D(L) = \left\lfloor \frac{L + D_i - C_i}{T_i} \right\rfloor \tag{3}$$

Making use of D_k^* and C_k^* to account for the fact that task τ_k is schedulable under gFPDS if it is able to start its FNR by D_k^* results in the following schedulability test:

Deadline Analysis (DA) test for gFPDS: *A sporadic taskset is schedulable, if for every task τ_k, inequality (4) holds.*

$$D_k^* \geq C_k^* + \left\lfloor \frac{1}{m} \left(\sum_{\forall i \in hp(k)} I_i^D(D_k^*, C_k^*) + \sum_{\forall i \in lpv(k)} I_i^D(D_k^*, C_k^*) \right) \right\rfloor \tag{4}$$

where $lpv(k)$ is the set of virtual tasks used to model the non-pre-emptive execution of tasks in $lp(k)$. (Note the floor function comes from the use of integer values for all task parameters, and the fact that all m processor must be busy for at least $D_k^* - C_k^* + 1$ if they are to prevent task τ_k from executing for time C_k^*).

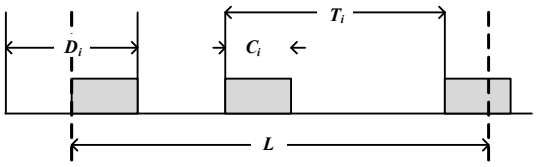

Figure 2: DA analysis: Interference within an interval.

With the DA test for gFPDS, the effect of push-through blocking from the FNR of the previous job of task τ_k is factored into the interference term for higher priority tasks. This is the case because the first (carry-in) job of each higher priority task τ_i within the interval of interest is assumed to execute as *late as possible*, i.e. just before its deadline, and then subsequent jobs of τ_i are assumed to execute as early as possible,

[2] *Push-through blocking* is the term used to describe the situation where the non-pre-emptive behaviour of one job of a task delays some higher priority job that subsequently interferes with the execution of the next job of the task.

see Figure 2. Provided that each higher priority task τ_i is itself schedulable then this accounts for any push-through blocking effect from the FNR of the previous job of task τ_k. This is because task τ_k has a constrained deadline, and thus the effect of push-through blocking can only be via a delay in the execution of one or more higher priority jobs, and such jobs are already assumed to be able to incur the maximum possible delay.

We now extend the DA test using the approach of [Guan et al. 2009]. They showed that for gFPPS, an upper bound on the interference over an interval L due to a higher priority task τ_i *without* a carry in job is given by:

$$I_i^{NC}(L,C) = \min(W_i^{NC}(L), L - C + 1) \tag{5}$$

where:

$$W_i^{NC}(L) = N_i^{NC}(L)C_i + \min(C_i, L - N_i^{NC}(L)T_i) \tag{6}$$

and

$$N_i^{NC}(L) = \lfloor L / T_i \rfloor \tag{7}$$

The difference between the interference terms (1) and (5) is:

$$I_i^{DIFF-D}(L,C) = I_i^{D}(L,C) - I_i^{NC}(L,C) \tag{8}$$

[Davis and Burns 2011a] showed that the worst-case scenario for gFPPS occurs when there are at most m-1 carry-in jobs. We now prove that for gFPDS, we similarly only need to consider interference from at most m-1 higher priority tasks with carry-in jobs. The proof follows closely the approach of [Guan et al. 2011] for non-pre-emptive fixed priority scheduling, with suitable adaptations for pre-emptive tasks with a FNR.

The proof uses the concept of a *problem window*, which relates to a job of task τ_k missing its deadline. Let J_k be the first job of τ_k that misses its deadline, with r_k being the release time of that job and d_k its absolute deadline. Further, let $l_k = d_k - (F_k - 1)$ be the latest time by which the job must have started its FNR in order to complete execution by its deadline (see Figure 3).

We use $\Psi(t,k)$ to denote the set of tasks with higher priority than τ_k (i.e. $hp(k)$) that have active jobs (i.e. jobs that have been released but not yet completed) at time t. Let t_0 be the earliest time before the release of the problem job J_k at r_k such that $\forall t \in [t_0, r_k)$ at least one of the following holds:

(i) $|\Psi(t,k)| = m$ and all of the tasks in $\Psi(t,k)$ execute in the interval $[t, t+1)$.

(ii) There are some tasks in $\Psi(t,k)$ that do not execute in the interval $[t, t+1)$.

If there is no such t_0, then $t_0 = r_k$. The start, length, and end of the problem window are defined as follows: start at t_0, length $L = D_k - (F_k - 1)$, end at $t_f = t_0 + L$. Note the length of the problem window is fixed, but its start and end points are not.

We make the pessimistic assumption that the final non-pre-emptive regions of tasks of priority lower than k, are represented by virtual tasks $lpv(k)$ with the highest priority. This and non-pre-emptive execution of the FNR of previous jobs of task τ_k is why case (ii) can occur. From the definition of t_0, we note that no job of a task with priority k or lower can start to execute during the interval $[t_0, r_k)$.

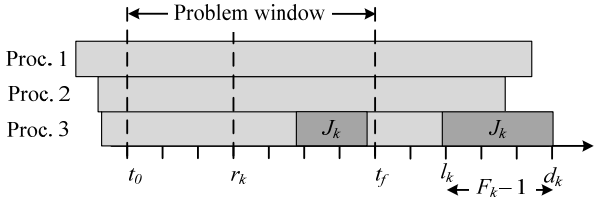

Figure 3: Problem window.

LEMMA 1: At most m-1 tasks in $hp(k)$ have carry-in jobs i.e., jobs that were released strictly prior to the start of the problem window, but have outstanding execution at the start of it.

PROOF.: At time $t_0 - 1$ then given how t_0 is defined, it follows that the number of higher priority tasks with active jobs cannot be m ($|\Psi(t_0 - 1, k)| \neq m$) and *all* of those jobs must be executing in the interval $[t_0 - 1, t_0)$. As no more than m tasks can have jobs that are executing at the same time then it follows that $|\Psi(t_0 - 1, k)| < m$. The number of higher priority tasks in $hp(k)$ with carry-in jobs is therefore limited to a maximum of m-1 □

LEMMA 2: The maximum interference from all previous jobs (prior to J_k) of task τ_k in the problem window is $(F_k - 1)$.

PROOF. Let J_k^- (released at time r_k^-) be the previous job of task τ_k to J_k. We consider two cases:

Case 1: $t_0 \leq r_k^-$: Since by the definition of t_0 no job of priority k or lower is able to start executing in the interval $[t_0, r_k)$ it follows that J_k^- is unable to start execution until time $r_k \geq r_k^- + T_k$ which as $D_k \leq T_k$ means that J_k^- misses its deadline. However, this contradicts the fact that J_k is the first job of task τ_k to miss a deadline, hence it cannot be the case that $t_0 \leq r_k^-$.

Case 2: $t_0 > r_k^-$: By the definition of t_0, job J_k^- can only execute from t_0 onwards if it has already entered its final non-pre-emptive region. Hence the maximum interference from J_k^- in the problem window is $(F_k - 1)$ □

We now form the DA-LC sufficient schedulability test for task τ_k by determining if job J_k can be guaranteed to start its final non-pre-emptive region by the end of the problem window. To simplify the analysis, we first prove the following lemma.

LEMMA 3: Consider an alternative scenario which is identical to the original (depicted in Figure 3) except that the problem job J_k is assumed to be released at time t_0 (rather than at time r_k) and have a deadline at $t_0 + D_k$, without implication on the release times or deadlines of the previous jobs of task τ_k. If job J_k is schedulable in this alternative scenario, then it is also schedulable in the original scenario.

PROOF: By the definition of t_0, no work of priority k or lower can start in the interval $[t_0, r_k)$. It follows that in the alternative scenario, job J_k cannot begin executing until at least time r_k, hence the job executes in exactly the same time intervals in the alternative scenario as it does when released at r_k in the original scenario. As the deadline of J_k is no later in the alternative scenario (i.e. $t_0 + D_k \leq d_k$), schedulability of job J_k in the alternative scenario implies schedulability of J_k in the original scenario □

We construct the DA-LC test based on the alternative scenario. We assume a problem window starting at t_0 and having a fixed length of $L = D_k^* = D_k - (F_k - 1)$, since we are interested in whether job J_k can start its FNR by completing execution $C = C_k^* = C_k - (F_k - 1)$ within this time interval, and hence meet its deadline. The worst-case interference in the problem window (of length $L = D_k^* = D_k - (F_k - 1)$) due to a higher priority task τ_k with a carry-in job is given by (1) and without a carry-in job it is given by (5), assuming that $C = C_k^* = C_k - (F_k - 1)$. Due to Lemma 1 we include interference from at most m-1 higher priority tasks with carry-in jobs. Further, by Lemma 2 we include only $(F_k - 1)$ as push-through blocking from previous jobs of task τ_k. Finally, as the FNR's of lower priority tasks can be entered either prior to the problem window or at any time during it when job J_k is executing, we include these FNRs as virtual tasks of higher priority, with carry-in jobs. Together this gives the formulation of the DA-LC test presented in (9). Since this test is valid for the alternative scenario described in Lemma 3, it is also valid for the original scenario.

Effectively, the test builds directly upon the DA-LC analysis for gFPPS [Davis and Burns 2011a] by adding terms that upper bound the additional interference that could potentially occur during the worst-case problem window of task τ_k due to the non-pre-emptive execution of the FNRs of tasks of lower priority than τ_k as well as due to push-through blocking from previous jobs of task τ_k itself, thus modelling this execution as if it were due to higher priority tasks.

Deadline Analysis – Limited Carry-in (DA-LC test) for gFPDS: *A sporadic taskset is schedulable, if for every task τ_k, inequality (9) holds:*

$$D_k^* \geq C_k^* + \left\lfloor \frac{1}{m} \left(\sum_{\forall i \in hp(k)} I_i^{NC}(D_k^*, C_k^*) + \sum_{i \in MD(k,m-1)} I_i^{DIFF-D}(D_k^*, C_k^*) + \sum_{\forall j \in lpv(k)} I_j^{D}(D_k^*, C_k^*) + F_k - 1 \right) \right\rfloor \tag{9}$$

where $MD(k, m\text{-}1)$ is the subset of at most m-1 tasks with the largest values of $I_i^{DIFF-D}(D_k^*, C_k^*)$ from $hp(k)$, $lpv(k)$ is the set of virtual tasks used to model the non-pre-emptive execution of tasks in $lp(k)$, and the final $F_k - 1$ term accounts for the effects of push-through blocking from the FNR of the previous job of task τ_k.

With the DA-LC schedulability test, as we limit the number of higher priority tasks with carry-in jobs to at most $m - 1$, the effect of push-through blocking from the previous job of task τ_k is not necessarily accounted for within the interference terms for higher priority tasks. This can be seen by considering the degenerate case of a single processor ($m = 1$). Here, $m - 1 = 0$, which implies that no higher priority tasks have carry-in jobs, and hence the interference terms for these tasks do not account for push-through blocking. This point was not recognised in the preliminary version of this paper published in RTCSA [Davis et al. 2013]. Here, we correct this omission by including the separate term $F_k - 1$ in (9) to account for the push-through blocking effect of the FNR of the previous job of task τ_k. Note Lemma 2 shows that only a single value is needed here. (The counterexample given in Table 2 of [Davis et al. 2007], assuming $m = 1$, is sufficient to show that this term is necessary and without it, the DA-LC test can potentially give optimistic results).

4.2 Response Time Analysis for gFPDS

We now extend and adapt the response time test of [Bertogna and Cirinei 2007] to gFPDS. They showed that under gFPPS, if task τ_k is schedulable in an interval of length L, completing an execution time of C time units, then an upper bound on the interference in that interval due to a higher priority task τ_i with a carry-in job is given by the following equation. (Note the R superscript denotes *Response Time Analysis*).

$$I_i^R(L, C) = \min(W_i^R(L), L - C + 1) \tag{10}$$

where, $W_i^R(L)$ is an upper bound on the workload of task τ_i in an interval of length L, taking into account the upper bound response time R_i^{UB} of task τ_i (see Figure 4):

$$W_i^R(L) = N_i^R(L) C_i + \min(C_i, L + R_i^{UB} - C_i - N_i^R(L) T_i) \tag{11}$$

where $N_i^R(L)$ is given by:

$$N_i^R(L) = \left\lfloor \frac{L + R_i^{UB} - C_i}{T_i} \right\rfloor \tag{12}$$

Making use of D_k^* and C_k^* to account for the fact that task τ_k is schedulable under gFPDS if it is able to start its FNR by D_k^* results in the following schedulability test. (Note, we return later to the order in which upper bound response times are computed, which is resolved by Algorithm 1).

Response Time Analysis (RTA) test for gFPDS: *A sporadic taskset is schedulable, if for every task τ_k, the upper bound response time R_k^S for the start (first unit of execution) of the task's FNR, computed via the fixed point iteration given by (13) within Algorithm 1, is less than or equal to the task's effective deadline D_k^*:*

$$R_k^S \leftarrow C_k^* + \left\lfloor \frac{1}{m} \left(\sum_{\forall i \in hp(k)} I_i^R(R_k^S, C_k^*) + \sum_{\forall i \in lpv(k)} I_i^R(R_k^S, C_k^*) \right) \right\rfloor \quad (13)$$

In (13), the second summation term models the blocking effect from lower priority tasks via the set of virtual tasks. If task τ_k is schedulable, then $R_k^{UB} = R_k^S + (F_k - 1)$.

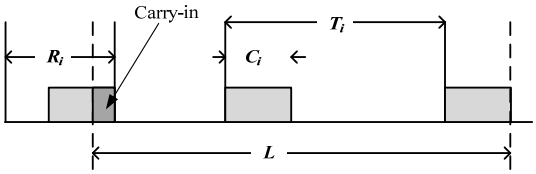

Figure 4: RTA analysis: Interference within an interval.

With the RTA test for gFPDS, the effect of push-through blocking from the FNR of the previous job of task τ_k is factored into the interference term for higher priority tasks. This is the case because the first (carry-in) job of each higher priority task τ_i within the interval of interest is assumed to execute as *late as possible*, in this case just before its worst-case response time, and then subsequent jobs of τ_i are assumed to execute as early as possible, see Figure 4. Provided that each higher priority task τ_i is itself schedulable then this is sufficient to account for any push-through blocking effect from the FNR of the previous job of task τ_k. This is because task τ_k has a constrained deadline, and thus the effect of push-through blocking can only be via a delay in the execution of one or more jobs of higher priority tasks, and the worst-case response times of these tasks are computed (highest priority first) assuming the effects of interference from the virtual tasks representing the FNRs of all lower priority tasks, including task τ_k.

We now extend the RTA test using the approach of [Guan et al. 2009]. They showed that under gFPPS, if a higher priority task τ_i does not have a carry-in job, then the interference term is given by (5) rather than (10). The difference between the two interference terms is:

$$I_i^{DIFF-R}(L,C) = I_i^R(L,C) - I_i^{NC}(L,C) \quad (14)$$

In the RTA test formulation, the length of the problem window is variable; however, this does not affect the validity of Lemmas 1-3. Hence, to form a more sophisticated test, we limit the interference considered from higher priority tasks with carry-in jobs to at most m-1 such tasks, with $(F_k - 1)$ as push-through blocking from previous jobs of task τ_k. Thus an improved test for gFPDS is as follows:

Response Time Analysis – Limited Carry-in (RTA-LC) test for gFPDS: *A sporadic taskset is schedulable, if for every task τ_k, the upper bound response time R_k^S for the start (first unit of execution) of the task's FNR, computed via the fixed point iteration given by (15) within Algorithm 1, is less than or equal to the task's effective deadline D_k^*:*

$$R_k^S \leftarrow C_k^* + \left\lfloor \frac{1}{m} \left(\sum_{\forall i \in hp(k)} I_i^{NC}(R_k^S, C_k^*) + \sum_{i \in MR(k,m-1)} I_i^{DIFF-R}(R_k^S, C_k^*) + \sum_{\forall j \in lpv(k)} I_j^R(R_k^S, C_k^*) + F_k - 1 \right) \right\rfloor \quad (15)$$

where $MR(k, m\text{-}1)$ is the subset of at most m-1 tasks with the largest values of $I_i^{DIFF-R}(R_k^{UB}, C_k)$, given by (14), from the set of tasks $hp(k)$, $lpv(k)$ is the set of virtual tasks used to model the non-pre-emptive execution of tasks in $lp(k)$, and the final $F_k - 1$ term accounts for the effects of push-through blocking from the FNR of the previous job of task τ_k.

If task τ_k is schedulable, then $R_k^{UB} = R_k^S + (F_k - 1)$.

Similar to the case with the DA-LC test, with the RTA-LC test, because we limit the number of higher priority tasks with carry-in jobs to at most $m - 1$, the effect of push-through blocking from the previous job of task τ_k is not necessarily accounted for within the interference terms for higher priority tasks. We therefore include the final $F_k - 1$ term to account for this, correcting the formulation of the RTA-LC test given in the preliminary version of this paper published in RTCSA [Davis et al. 2013].

ALGORITHM 1. Response Time Iteration

```
1   Initialize all R_i^UB = C_i
2   repeat = true
3   while (repeat) {
4       repeat = false
5       for (each priority level k, highest first) {
6           Calc. R_k^UB via RTA or RTA-LC test for gFPDS
7           if ( R_k^UB > D_k ) {
8               Return unschedulable
9           }
10          if ( R_k^UB differs from its previous value) {
11              repeat = true
12          }
13      }
14  }
15  return schedulable
```

We note that in adapting the methods of [Bertogna and Cirinei 2007] and [Guan et al. 2009] to gFPDS there is a difficulty in accounting for the interference from virtual tasks. When computing the upper bound response time for task τ_k the upper bound response times of each higher priority task are required. This can easily be achieved for the set of tasks $hp(k)$ simply by computing response times in order, highest priority first, which is all that is needed for gFPPS. However, when considering gFPDS we also include interference from virtual tasks corresponding to tasks in $lp(k)$. Here, the upper bound response time R_{iv}^{UB} for each virtual task equates to that of its corresponding (lower priority) task $R_{iv}^{UB} = R_i^{UB}$, which itself depends on the upper bound response time of task τ_k, leading to an apparent circularity. This would seem to imply that we cannot compute response times in either lowest priority first or highest priority first order. However, we note that this issue can be solved by adding an outer loop that forms a fixed point iteration. The pseudo code in Algorithm 1 implements this approach. First, all of the upper bound response times are initialised to a guaranteed lower bound on their values: $R_i^{UB} = C_i$ (line 1). Then new upper bound response times are computed for each task τ_k in order, highest priority first, (line 6). As $R_i^{UB} \ \ \forall i \in lp(k)$ have not yet been computed on the current iteration, this calculation uses the upper bound response times for virtual tasks (associated with tasks in $lp(k)$) from the previous iteration of the while loop (lines 3-14), with $R_{iv}^{UB} = R_i^{UB} = C_i$ used on the first iteration. Since the values of R_i^{UB} for some lower priority tasks may get larger when they are re-calculated, the algorithm iterates until there are no further changes in the response times (line 10) or a task is found that is unschedulable (line 7). Convergence (or exceeding a

deadline) is guaranteed due to the monotonic dependencies between response times: The upper bound response time R_{iv}^{UB} of each virtual task is monotonically non-decreasing with respect to increases in the upper bound response times of all tasks in $hp(i)$, and the upper bound response time R_k^{UB} of each task τ_k is monotonically non-decreasing with respect to increases in the upper bound response times of all virtual tasks associated with tasks in $lp(k)$.

4.3 Complexity and comparability

As part of the DA and DA-LC tests for gFPDS, the complexity of computing (4) or (9) is $O(n)$, as the (m-1) largest I_i^{DIFF} terms may be obtained by *linear-time selection* [Blum et al. 1973]. The DA and DA-LC tests are therefore polynomial in complexity: $O(n^2)$ for a taskset of cardinality n.

As part of the RTA and RTA-LC tests for gFPDS, the complexity of computing one iteration of (13) or (15) is $O(n)$. The response time calculation can take at most $D_k - C_k$ iterations for task τ_k since iteration starts with $R_k^{UB} = C_k$, and the task is deemed unschedulable if $R_k^{UB} > D_k$. Hence the complexity of computing (13) or (15) once for task τ_k is $O(nD_k)$. The complexity of the inner loop (lines 5-13) in Algorithm 1 is therefore $O(n^2 D_{\max})$, where $D_{\max}$ is the longest task deadline. Further, the outer loop (lines 3-14) of Algorithm 1 iterates at most D_{sum} times, where D_{sum} is the sum of task deadlines, since on each iteration some response time must increase by at least one for the loop to continue iterating. Hence the overall complexity of Algorithm 1 and the RTA and RTA-LC tests is $O(n^2 D_{\max} D_{sum})$.

The following comparability relationships hold between the various schedulability tests for gFPDS. The RTA test dominates the DA test, and the RTA-LC test dominates the DA-LC test. However, in contrast to the equivalent tests for gFPPS the RTA-LC and RTA tests for gFPDS are incomparable, as are the DA-LC and DA tests. This is due to the different ways in which push-through blocking is accounted for by these tests. We note that each test for gFPDS reduces to the corresponding test for gFPPS if all FNR lengths are set to 1; thus each schedulability test for gFPDS dominates its counterpart for gFPPS. This is the case because all tasksets deemed schedulable by a test for gFPPS are also schedulable according to the corresponding gFPDS test with FNR lengths set to 1, and there are also tasksets that are deemed schedulable by a gFPDS test (with some FNR lengths not equal to 1) that are not schedulable according to the corresponding gFPPS test, assuming fully pre-emptive behaviour.

4.4 Optimal priority assignment

[Davis and Burns 2009; 2011a] showed that Audsley's OPA algorithm [Audsely 1991; 2001] can be used to obtain an optimal priority assignment with respect to any schedulability test that fulfils the following three conditions:

Condition 1: The schedulability of a task τ_k may, according to test S, depend on the set of tasks with priorities higher than k, but not on their relative priority ordering.

Condition 2: The schedulability of a task τ_k may, according to test S, depend on the set of tasks with priorities lower than k, but not 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).

Inspection of the DA and DA-LC tests for gFPDS shows that these conditions hold (assuming fixed values of F_i) and so these tests are OPA-compatible. Whereas the dependency on the upper bound response time R_i^{UB} of higher priority tasks in (11) means that the RTA and RTA-LC tests are not OPA-compatible, since they violate Condition 1.

4.5 Example of gFPDS

We now provide an example comparing gFPDS with gFPPS and gFPNS. The example is based on the taskset in TABLE I. This taskset is trivially unschedulable on two processors with any form of fixed priority scheduling unless task τ_C has the lowest priority. Since task τ_A and task τ_B are equivalent, placing either of them at the lowest priority would make that task have a response time of 6 and so be unschedulable. Thus, there is only one viable priority ordering: τ_A, τ_B, τ_C .

Table I: Task parameters

Task	Execution time	Period	Deadline
τ_A	3	10	5
τ_B	3	10	5
τ_C	8	25	12

With pre-emptive scheduling (gFPPS), if tasks τ_A and τ_B are released simultaneously, then task τ_C misses its deadline, as shown in Figure 5(a).
Similarly, with non-pre-emptive scheduling (gFPNS), if task τ_C is released just before tasks τ_A and τ_B, then task τ_B misses its deadline.

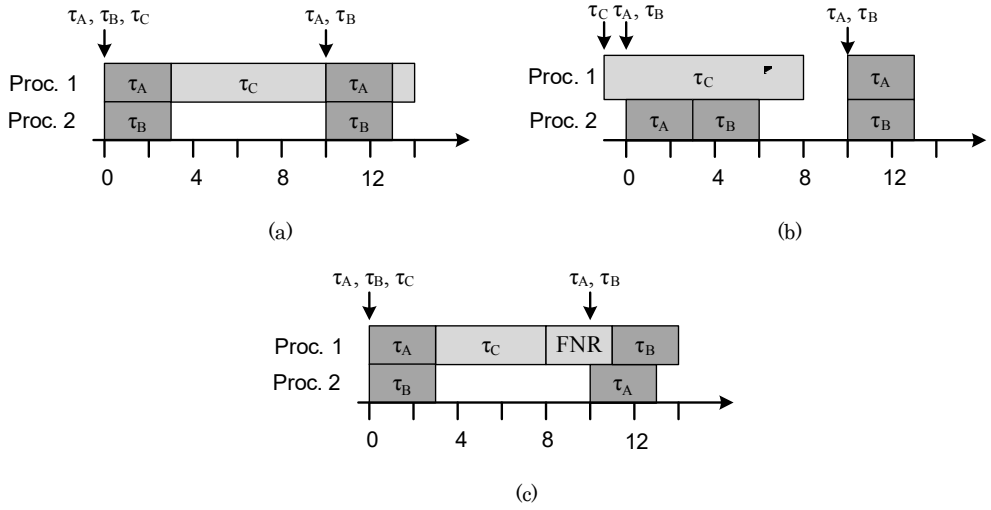

Figure 5: Schedule with (a) gFPPS (b) gFPNS and (c) gFPDS.

However, if we use deferred pre-emption and let $F_A = 1, F_B = 1$, and $F_C = 3$, then using the RTA test, we obtain $R_1^{UB} = 3$, $R_2^{UB} = 5$, and $R_3^{UB} = 11$; proving that the taskset is schedulable. Here, the FNR of task τ_C is enough to ensure that there can be no second pre-emption by task τ_A, yet task τ_C only blocks tasks τ_A and τ_B for a maximum of 2 time units enabling their deadlines to be met. This example illustrates the strict dominance, rather than equivalence, of gFPDS over gFPPS and gFPNS.

Note, this example has been deliberately constructed with Deadline Monotonic Priority Ordering (DMPO) as the only feasible priority ordering; however, it is well

known that DMPO is not optimal for global fixed priority scheduling, and is not even a good heuristic [Davis and Burns 2009; 2011a].

5. OPTIMAL GFPDS

In this section, we build upon the ideas and techniques developed in [Davis and Bertogna 2012] which provide optimal algorithms for fixed priority scheduling with deferred pre-emption for uniprocessor systems. We pose the same two problems relating to the assignment of FNR lengths and priorities for the multiprocessor case, i.e. under gFPDS. We show that the first of these problems can be solved in a similar way to the uniprocessor case, and via a counterexample, that the second problem cannot.

Problem 1: *Final Non-pre-emptive Region length Problem* (*FNR Problem*). For a given taskset complying with the task model described in Section 3, and a given priority ordering X, find a length for the FNR of each task such that the taskset is deemed schedulable under gFPDS by schedulability test S.

Definition 1: An algorithm A is said to be *optimal* for the *FNR Problem* with respect to a schedulability test S, if there are no taskset / priority assignment combinations that are deemed schedulable under gFPDS by test S with some set of FNR lengths, that are not also deemed schedulable by the test using the set of FNR lengths determined by algorithm A.

Problem 2: *Final Non-pre-emptive Region Length and Priority Assignment Problem* (*FNR-PA Problem*). For a given taskset complying with the task model described in Section 3, find both (i) a priority assignment, and (ii) a set of FNR lengths that makes the taskset schedulable under gFPDS according to schedulability test S.

Definition 2: An algorithm B is said to be *optimal* for the *FNR-PA Problem* with respect to a schedulability test S, if there are no tasksets compliant with the task model that are deemed schedulable under gFPDS by test S with some priority assignment X and some set of FNR lengths, that are not also deemed schedulable using the priority assignment and set of FNR lengths determined by algorithm B.

5.1 Sustainability with respect to FNR lengths

In order to be able to solve Problems 1 and 2 efficiently, we would prefer to use schedulability tests that are sustainable [Baruah and Burns 2006; Burns and Baruah 2008] with respect to changes in the length of a task's FNR. With a sustainable test, we can use binary search to help solve the problems. In contrast with an unsustainable test, we would potentially need to check every possible value for the FNR length of each task which is typically not practical without some form of approximation.

THEOREM 1: The DA schedulability test for task τ_k under gFPDS is sustainable with respect to *increases* in the length F_k of the task's FNR.

PROOF: To prove the theorem, it suffices to show that if (4) holds for some pair of values (C_k^*, D_k^*), then it continues to hold for the pair of values $(C_k^* - z, D_k^* - z)$ where z is a positive integer ($z \le C_k^*$). Substituting $C_k^* - z$ for C_k^* and $D_k^* - z$ for D_k^* in (4), we need to show that the summation terms do not increase. By inspecting the component equations (1) – (3), we observe that the interference within a window of length L is monotonically non-decreasing with respect to the length of the window (i.e. it is no larger for an interval of length $D_k^* - z$ than it is for an interval of length D_k^*). Further, we must also consider the dependence of component equation (1) on C. C appears in the expression $L - C + 1$. which is unchanged by subtracting z from both

L and C. The summation terms in (4) are therefore monotonically non-increasing with respect to increasing values of z □

COROLLARY 1: The schedulability of a task is, according to the DA test, a monotonically non-decreasing function of the length of its FNR.

THEOREM 2: (Negative result) The RTA schedulability test for task τ_k under gFPDS is *not sustainable* [Baruah and Burns 2006; Burns and Baruah 2008] with respect to increases in the length F_k of the task's FNR.

PROOF: Increasing the FNR length F_k of task τ_k increases the execution time of its associated virtual task τ_{kv} (as $C_{kv} = F_k - 1$). With the RTA test this can result in a large increase in the upper bound response time R_i^{UB} of some higher priority task τ_i due to the inclusion of interference from an extra job of a yet higher priority task, as well as the extra interference from τ_{kv} (i.e. blocking). The increase in R_i^{UB} can cause an extra job of task τ_i to interfere in the problem window of task τ_k making it unschedulable according to the test.

This scenario occurs with the taskset described in Table II below, assuming two processors. In this case, if task τ_D is fully pre-emptive, then the computed upper bound response times are 10, 5, 10 and 23 for tasks τ_A, τ_B, τ_C, and τ_D respectively; however, increasing the FNR length of task τ_D so that $F_D = 2$ results in upper bound response times of 10, 6, 15, and 27, which would make task τ_D unschedulable if it had a deadline of 25. This increase in the upper bound response time of task τ_D is due to the large increase in the upper bound response time of task τ_C from 10 to 15, and the subsequent inclusion of an extra job of task τ_C in the problem window of task τ_D. It is easy to construct examples where decreasing the FNR length of a task τ_k can result in the task becoming unschedulable due to additional pre-emptions from higher priority tasks □

TABLE II: EXAMPLE TASK PARAMETERS

Task	Execution time	Period	Deadline
τ_A	10	100	10
τ_B	5	10	10
τ_C	5	15	15
τ_D	7	100	100

THEOREM 3: (Negative result) The DA-LC and RTA-LC schedulability tests for task τ_k under gFPDS are not sustainable [Baruah and Burns 2006; Burns and Baruah 2008] with respect to increases in the length F_k of the task's FNR.

PROOF: Increasing the FNR length F_k of task τ_k increases the $F_k - 1$ term in (9) and (15), while reducing both D_k^* and C_k^* by the same amount. As it is possible for the summation terms in (9) and (15) to be unaffected by this change (for example if all of the tasks have long periods), then the increase in F_k can result in task τ_k being deemed unschedulable by the test when it was previously deemed schedulable with a shorter FNR length. This can be trivially seen by considering the degenerate case of a single processor, and a single task τ_k with $C_k = D_k$. Any increase in F_k above 1 (i.e. the fully pre-emptive case) would make the task unschedulable according to the DA-LC and RTA-LC schedulability tests. This happens because the FNR is (pessimistically in this case) included in the response time twice, once as the final execution of task τ_k, in $R_k^{UB} = R_k^S + (F_k - 1)$, and once to account for push-through blocking, in (9) and (15). Hence we obtain $R_k^{UB} = C_k + (F_k - 1) > D_k$ for $F_k > 1$ and $R_k^{UB} = C_k = D_k$ for $F_k = 1$ □

5.2 Solving the FNR and FNR-PA Problems

To aid in solving the FNR and FNR-PA problems, we introduce the concept of a *blocking vector*. For a given taskset and priority ordering X, we use $B(k)$ to represent

the blocking vector at priority k, where the blocking vector relates to the set of FNR lengths of the ordered set of lower priority tasks $lp(k)$. Hence:

$$B(k) = ((F_n - 1),(F_{n-1} - 1)...(F_{k+1} - 1)) \quad (16)$$

We define a 'greater than or equal to' ($\geq$) and similarly a 'less than or equal to' ($\leq$) relationship between blocking vectors with the meaning $B^1 \geq B^2$ if every element in B^2 is no larger than the corresponding element in B^1.

THEOREM 4: Task schedulability under gFPDS according to the DA, DA-LC, RTA, or RTA-LC test is sustainable with respect to decreases in the blocking vector. Stated otherwise, according to the DA, DA-LC, RTA, or RTA-LC test, a task that is schedulable at priority k with a blocking vector $B(k)$ remains schedulable when the blocking vector is reduced (e.g. by reducing the FNR length of one or more lower priority tasks) and the sets $lp(k)$ and $hp(k)$ of lower and higher priority tasks remain unchanged.

PROOF: Follows directly from inspection of (4), (9), (13), and (15). In each case, reductions in the summation term over the set of virtual tasks can only improve schedulability □

COROLLARY 3: Using the DA, DA-LC, RTA, or RTA-LC schedulability test for gFPDS, the minimum schedulable FNR length F_k for a task τ_k is monotonically non-increasing with respect to decreases in the blocking vector. Stated otherwise, a smaller blocking vector at priority k cannot result in a larger minimum length for the FNR of the task at that priority level.

We now investigate using the FNR and FNR-PA algorithms presented by [Davis and Bertogna 2012] to solve Problems 1 and 2 for multiprocessor systems. The two algorithms are the same as those used in the uniprocessor case with the exception that the schedulability tests used are the DA or DA-LC tests for gFPDS. (The RTA and RTA-LC tests cannot be used here as the FNR and FNR-PA algorithms require that task schedulability is determined lowest priority first). Theorem 1 shows that in the case of the DA test, a binary search can be employed to determine the smallest FNR length commensurate with task schedulability. By contrast, Theorem 3 shows that in the case of the DA-LC test, a binary search cannot be used. Instead, the smallest schedulable FNR length for each task must be searched for by checking each possible value, smallest first. We return to this point in Section 7.

The proof of Theorem 5 uses the techniques from the uniprocessor case with minor adjustments for the way in which lower priority tasks now impinge on the schedulability of higher priority tasks.

ALGORITHM 2. FNR Algorithm

```
for each priority level k, lowest first {
    determine the smallest value for the final non-pre-emptive region length F(k) such that
    the task at priority k is schedulable according to test S. Set the length of the final non-pre-
    emptive region of the task to this value.
}
```

THEOREM 5: The FNR algorithm (Algorithm 2) is optimal for the FNR problem (see Problem 1 and Definition 1).

PROOF: We assume (for contradiction) that there exists a taskset τ and priority ordering X that is schedulable according to schedulability test S (either the DA or the DA-LC test for gFPDS), with some set of FNR lengths F'_k for k = 1 to n, and that the FNR algorithm fails to determine a set of FNR lengths F_k for k = 1 to n, that results in the taskset being schedulable according to the test.

Let $B'(k)$ be the blocking vector at priority k with the schedulable set of FNR lengths, and $B(k)$ be the blocking vector at priority k with the set of FNR lengths

computed by the FNR Algorithm. At each priority level, we will show that $F_k \leq F'_k$ and hence that $B(k) \leq B'(k)$ thus proving via Corollary 2 *sustainability of task schedulability with respect to blocking vectors* that the taskset is schedulable according to test S, with priority ordering X and the FNR lengths determined by the FNR Algorithm, thus contradicting the original assumption. The proof is by induction over each priority level k from n to 1.

Initial step: At the lowest priority level n, trivially we have $B(n) = B'(n) = \phi$. At priority n, the FNR Algorithm (Algorithm 2) computes, according to test S, the minimum schedulable FNR length F_n for task τ_n hence $F_n \leq F'_n$.

Inductive step: We assume that at priority k, $B(k) \leq B'(k)$ and $F_k \leq F'_k$, hence $B(k-1) \leq B'(k-1)$ and thus via Corollary 3, $F_{k-1} \leq F'_{k-1}$

Iterating over all of the priority levels shows that for all k from n to 1, $B(k) \leq B'(k)$ and so by Corollary 2, the taskset is schedulable, according to test S, with the set of FNR lengths F_k obtained by Algorithm 2 □

COROLLARY 4: (Follows from the proof of Theorem 5). For a given taskset and fixed priority ordering X, that is schedulable according to the DA or DA-LC schedulability test under gFPDS with some set of FNR lengths, Algorithm 2 minimises the FNR length of every task, and hence minimises the blocking vector at every priority level.

In contrast to the FNR problem, the FNR-PA problem requires a schedulable priority ordering to be established as part of the solution to the problem. Algorithm 3 which provides a solution to the FNR-PA problem in the uniprocessor case is based on Audsley's Optimal Priority Assignment (OPA) algorithm and uses a greedy bottom up approach.

THEOREM 6: (Negative result) The Final Non-pre-emptive Region Priority Assignment (FNR-PA) algorithm (Algorithm 3) is *not optimal* for the FNR-PA problem (see Problem 2 and Definition 2) in the multiprocessor case i.e. gFPDS using the DA or DA-LC schedulability tests.

PROOF: Proof is via a counterexample where the FNR-PA algorithm fails to find a schedulable combination of priority assignment and FNR lengths, when such a combination exists. The example is for the DA test, similar tasksets can be constructed for the DA-LC test. We assume a system with two processors and the taskset given in Table III. With four tasks, there are 24 distinct priority orderings ($n!$ = 24); however, in this case only two are schedulable, according to the DA test, given appropriate choices of FNR lengths.

Table III: Counterexample task parameters

Task	Execution time	Period	Deadline
τ_A	36	207	110
τ_B	86	178	141
τ_C	93	525	195
τ_D	62	767	195

First, we consider the behaviour of the FNR-PA algorithm. Starting at the lowest priority level, the FNR-PA algorithm checks each task in turn and finds the following: tasks τ_A and τ_B are not schedulable at priority 4 irrespective of FNR lengths; task τ_C is schedulable with a minimum FNR length of $F_C = 58$; and task τ_D is schedulable with a minimum FNR length of $F_D = 42$. Hence the FNR-PA algorithm chooses task τ_D and assigns it priority 4. Next schedulability of the remaining unassigned tasks is considered at priority 3. Tasks τ_A and τ_B are again not schedulable irrespective of FNR lengths; however, task τ_C is schedulable with a minimum FNR length of $F_C = 38$, hence it is assigned priority 3. Priority level 2 is then considered. Here, due to the large combined blocking effect modelled as the

virtual tasks τ_{Dv} and τ_{Cv} (i.e. 41 + 37 = 78) neither task τ_A nor τ_B is schedulable with any valid FNR length. The FNR-PA algorithm therefore declares the taskset unschedulable.

To prove the theorem, it now suffices to show that there is a priority and FNR length assignment where this taskset is schedulable according to the DA test. If we assign task τ_C the lowest priority, then it requires a minimum FNR length of $F_C = 58$ to be schedulable. Again we find that tasks τ_A and τ_B are not schedulable at priority 3; however, assigning task τ_D to priority 3, we find that it is schedulable with $F_D = 1$ (i.e. fully pre-emptive). Now, the blocking effect on whichever task, τ_A or τ_B, we choose for priority 2 is only 57, and hence either task is schedulable at that priority with the other task at priority 1. In both cases we have $F_A = 1$ and $F_B = 1$. Hence (τ_A, τ_B, τ_D, τ_C) and (τ_B, τ_A, τ_D, τ_C) are both schedulable priority orderings with FNR lengths of (1, 1, 1, 58).

Since the FNR-PA algorithm fails to find a schedulable combination of priority ordering and FNR lengths even though such a schedulable combination exists, this taskset provides a counterexample to the optimality of the algorithm □

ALGORITHM 3 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, according to test S 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
```

We note that the optimality of the FNR-PA algorithm breaks down in the multiprocessor case, because the blocking effect depends on a summation over the FNR lengths of lower priority tasks rather than a maximum, as in the single processor case. Minimising the FNR length at a given priority level does not necessarily minimise this summation, as shown in the above counterexample.

6. PARTITIONED FPDS (PFPDS)

In this short section, we briefly discuss partitioned, as opposed to global multiprocessor scheduling using final non-pre-emptive regions to improve schedulability.

[Davis and Bertogna 2012] introduced an optimal algorithm for fixed priority scheduling with deferred pre-emption on a single processor. This algorithm employs an exact schedulability test for FPDS and the FNR-PA algorithm (Algorithm 3) to find a schedulable priority assignment and set of FNR lengths whenever such a schedulable combination exists. Hence, for a multiprocessor system using partitioned scheduling, this technique yields the optimal assignment of priorities and FNR lengths for each single processor sub-problem, assuming that an allocation of tasks to processors has already been defined. Unfortunately, the allocation problem is

analogous to the bin packing problem and is known to be NP-Hard [Garey and Johnson 1979], thus heuristic task allocation schemes need to be employed.

Allocations schemes can be classified in terms of the bin packing approach used, which determines the order in which processors are chosen, e.g. First-Fit (FF), Next-Fit (NF) Best-Fit (BF), and Worst-Fit (WF). They can also be further categorised in terms of the order in which tasks are examined, e.g. decreasing density[3], decreasing deadline etc. For partitioned FPPS, allocation schemes based on First-Fit and either decreasing utilisation (or density) [Oh and Baker 1998] and decreasing deadline [Fisher et al. 2006] have proven effective.

By using FPDS and final non-pre-emptive regions, rather than fixed priority fully pre-emptive scheduling, then in the single processor case, we cannot improve upon the theoretical limits on performance obtained for FPPS [Davis et al. 2009] in terms of processor speedup factors [Kalyanasundaram and Pruhs 1995] (This is easily seen by considering the addition of a task with an infinitesimally small execution time, a very short deadline and long period to any system, which negates the possibility of effective FNRs). However, for more typical values for taskset parameters, partitioned FPDS may offer significant advantages over partitioned FPPS. We evaluate this possibility in the next section.

7. EXPERIMENTAL EVALUATION

In this section, we compare the performance of multiprocessor scheduling using global approaches: gFPDS, gFPPS, and gFPNS, and also partitioned approaches employing FPDS, FPPS, and FPNS on each processor.

For the global approaches, we compared the scheduling algorithms under the following priority assignment policies: (i) Deadline Monotonic (DMPO), (ii) DkC [Davis and Burns 2009; 2011a], and (iii) Audsley's Optimal Priority Assignment (OPA) algorithm for gFPPS and gFPNS. In the case of gFPDS, we used the FNR algorithm to obtain optimum final non-pre-emptive region lengths in conjunction with the heuristic priority assignment policies, and the FNR-PA Algorithm to provide both priority and FNR length assignment. For reference, we also made comparisons with FPZL [Davis and Burns 2011b; Davis and Kato 2012] which has some similarities in its behaviour to gFPDS, but belongs to the dynamic class of scheduling algorithms [Davis and Burns 2011c], which require substantially different RTOS support. The lines on the graphs are labelled according to the scheduling algorithm and priority assignment policy used, e.g. gFPDS (DkC). In all cases, we used the appropriate DA-LC test. Recall that in conjunction with this test for gFPDS, it is not possible to employ a binary search to find the smallest schedulable FNR of each task. Instead, we approximated checking all possible FNR lengths, smallest first, by examining 100 different FNR lengths for each task, with a granularity of $C_k/100$. We found that this approach, although approximate, provided significantly better performance than using the simpler DA test.

For the partitioned approaches, we employed optimal priority and FNR length assignment using the FNR-PA algorithm in the case of pFPDS, Audsley's Optimal Priority Assignment (OPA) algorithm for pFPNS, and Deadline Monotonic priority assignment for pFPPS. In each case we used First-Fit Decreasing Density (FFDD), First-Fit Decreasing Deadline (FFMaxD) , First-Fit Decreasing Execution Time (FFMaxC) as task allocation heuristics.

7.1 Parameter generation

The task parameters used in our experiments were randomly generated as follows:

[3] Where Density is defined as the WCET of a task divided by its deadline.

- First, an unbiased set of n utilisation values $U_i \leq 1$, were generated with a total utilisation of U, (see [Emberson et al. 2010] and [Davis and Burns 2011a] for how to generate an unbiased set of such values).
- Task periods were generated according to a log-uniform distribution (i.e. such that $\ln(T)$ has a uniform distribution). Here the ratio between the maximum and the minimum permissible task period was given by 10^r. By default, this range was 10, i.e. $r = 1$.
- Task execution times were set based on the task utilisation and period selected: $C_i = U_i T_i$.
- Task deadlines were *constrained* and chosen at random according to a uniform distribution in the range $[C_i + \alpha(T_i - C_i), T_i]$, with $\alpha = 0.5$ as the default.
- Taskset cardinality was z times the number of processors. By default, $z = 5$.

We examined systems with m = 2, 4, and 8 processors. In each experiment, the taskset utilisation was varied from $0.025m$ to $0.975m$ in steps of $0.025m$. For each utilisation value, 1000 tasksets were generated and their schedulability determined according to the various scheduling algorithms. Note due to the large number of lines on the graphs, the figures are best viewed online in colour.

7.2 Success ratio

In our first set of experiments, we compared the performance of the scheduling algorithms via the *success ratio*; the proportion of randomly generated tasksets that are deemed schedulable in each case.

Figure 6 shows the results of this experiment for the global scheduling algorithms and Figure 7 the corresponding results for the partitioned scheduling algorithms. The configuration used in each case, was a 4 processor system with a constrained deadline taskset of cardinality 20, and a range of task periods of 10.

For the global scheduling algorithms (Figure 6), we observe that the performance of gFPNS (dotted lines) was relatively poor for all priority assignment policies, due to the difficulty in accommodating tasks with long execution times and long deadlines. (With non-pre-emptive scheduling, such tasks cause significant blocking of higher priority tasks with short deadlines, leading to unschedulable systems at relatively modest utilisation levels). As expected, the results for gFPPS (dashed lines), show that optimal priority assignment outperformed the various heuristic priority assignment policies. Using gFPDS (solid lines with markers) substantially better results were obtained for the various heuristic priority assignment policies as compared to gFPPS, with the best performance obtained using the FNR-PA algorithm. In all cases, gFPDS significantly outperformed gFPPS and gFPNS assuming a like-for-like priority assignment policy. gFPDS using the FNR-PA algorithm resulted in performance part-way between that of gFPPS and the dynamic FPZL algorithm (solid line, no markers) assuming optimal priority assignment.

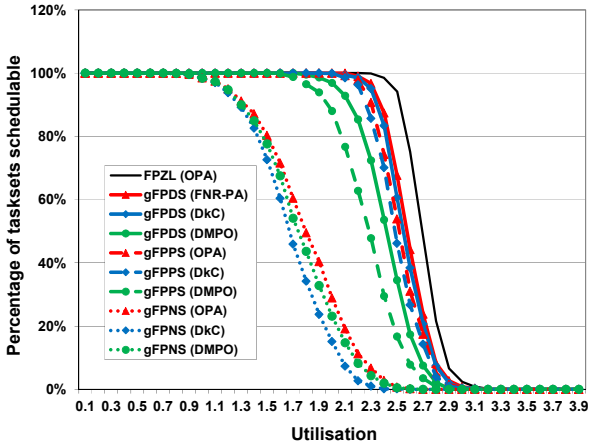

Figure 6: Success ratio: global scheduling algorithms $m = 4$, $n = 20$, r=1, constrained deadlines

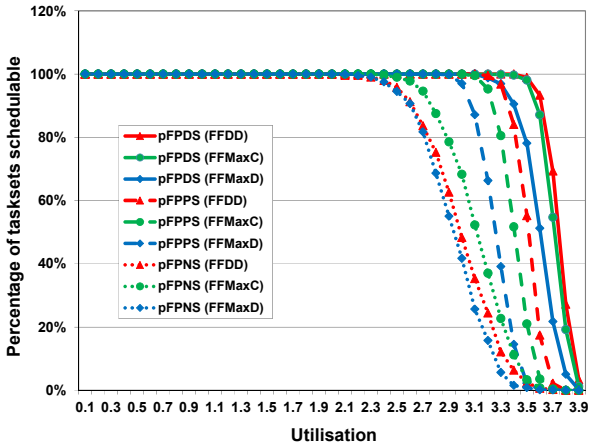

Figure 7: Success ratio: partitioned scheduling algorithms $m = 4$, $n = 20$, r=1, constrained deadlines

Figure 7 shows the results for the partitioned scheduling algorithms. Here, we observe that pFPDS (solid lines) provides significantly improved performance over pFPPS (dashed lines), with pFPNS providing the worst performance irrespective of the task allocation policy. Comparing the task allocation policies, First-Fit Decreasing Density (FFDD) has a clear advantage over First-Fit Decreasing Deadline (FFMaxD) in the case of pFPDS and pFPPS; and a small advantage over First-Fit Decreasing Execution Time (FFMaxC); however, this is reversed in the case of pFPNS. This reversal is due to the fact that allocating tasks according to their execution times has an advantage for non-pre-emptive scheduling, in that it tends to group tasks with large executions times (and long deadlines) together. This improves schedulability given that pre-emption is not permitted.

7.3 Weighted schedulability

In our second set of experiments we compared how the overall performance of each of the scheduling algorithms varies with respect to changes in a specific parameter via the *weighted schedulability measure* [Bastoni et al. 2010]. In the following figures we

show the weighted schedulability measure $W_S(p)$ for schedulability test S as a function of parameter p. For each value of p, this measure combines results for all of the tasksets generated for all of a set of equally spaced utilisation levels (e.g. 0.1 to 0.39 in steps of 0.1 for a 4 processor system). The schedulability test returns a binary result of 1 or 0 for each taskset τ and parameter p. Assuming that this result is given by $S(\tau, p)$ and $u(\tau)$ is the utilisation of taskset τ, then:

$$W_S(p) = \frac{\left(\sum_{\forall \tau} u(\tau) S(\tau, p)\right)}{\sum_{\forall \tau} u(\tau)} \tag{17}$$

The benefit of using the weighted schedulability measure is that it reduces a 3-dimensional plot to 2 dimensions, with individual results weighted by taskset utilisation to reflect the higher value placed on being able to schedule higher utilisation tasksets.

The first parameter examined was taskset cardinality. Figure 8 and Figure 9 show how the weighted schedulability varies with increasing taskset size (from 2 to 20 times the number of processors, i.e. from 4 to 80 tasks on an 4 processor system) for global and partitioned scheduling algorithms respectively.

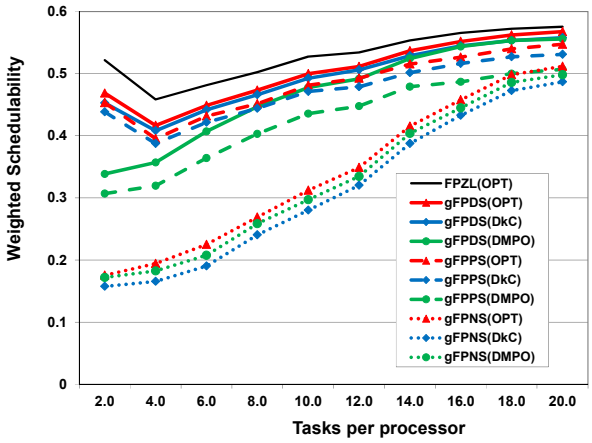

Figure 8: Weighted schedulability: global scheduling algorithms as a function of taskset size

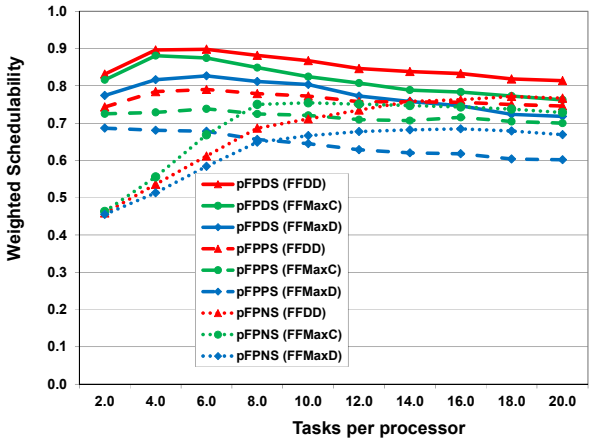

Figure 9: Weighted schedulability: partitioned scheduling algorithms as a function of taskset size

In Figure 8, we observe that increasing taskset cardinality results in tasks that have smaller utilisation on average and are therefore generally easier for global scheduling algorithms to schedule, as noted by [Davis and Burns 2009; 2011a]. For very low taskset cardinality, (e.g. twice as many tasks as processors), more tasksets are deemed schedulable by gFPPS and gFPDS. This happens because the pessimism in the tests for these global scheduling algorithms reduces with fewer tasks. This can be understood by considering Figure 2. Over any given interval, the interference assumed from two tasks with parameters $(C/2, D, T)$ can be more than, but is never less than that for a single task with parameters (C, D, T).

As the ratio of tasks to processors increases, the advantage conferred by deferred pre-emption gradually decreases. This is because the individual utilisation of each task is becoming quite small reducing the benefits that can be obtained over fully pre-emptive scheduling. The small size of the tasks also accounts for the improving performance of non-pre-emptive scheduling with an increased number of tasks.

Figure 9 shows how the weighted schedulability measure varies with taskset cardinality for the partitioned scheduling algorithms. Here it is clear that the deferred pre-emption approach (pFPDS) provides a significant advantage over both pFPPS and pFPNS. Further, this advantage is maintained as the number of tasks increases. We note that in this case, when the total number of tasks is very small (e.g. only twice as many tasks as processors), they become difficult to allocate to processors (bin-packing problem) due to their high utilisation, hence schedulability reduces. This is in direct contrast to global scheduling which improves in this case as discussed above.

The second parameter we examined was the range of task periods. Figure 10 and Figure 11 show how the weighted schedulability measure varies with the log-range r of task periods given by the ratio 10^r between the maximum and the minimum permissible task period, for the global and partitioned scheduling algorithms respectively. Here, the value of r was varied from r = 0.5 ($10^{0.5} = 3.16$) to r = 4 ($10^4 = 10{,}000$).

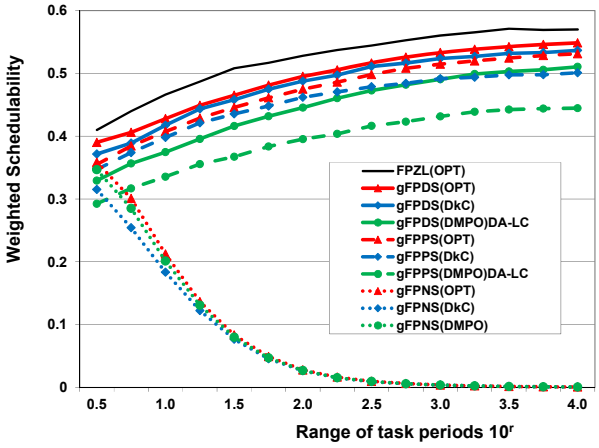

Figure 10: Weighted schedulability: global scheduling algorithms as a function of period range, D ≤ T

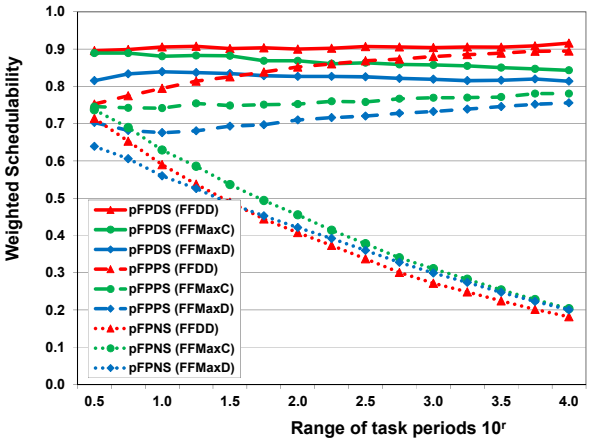

Figure 11: Weighted schedulability: partitioned scheduling algorithms as a function of period range, D ≤ T

For the global scheduling algorithms (Figure 10), we observe that gFPDS shows an improvement over gFPPS, which increases slightly when the range of task periods is relatively small. This is because with all task periods and deadlines of a similar duration, all of the tasks can typically tolerate significant blocking and so there is scope to choose FNR lengths that improve schedulability. As expected, both gFPPS and gFPDS show improved performance as the range of task periods increases, while gFPNS shows rapidly declining performance. This is because tasks with relatively long periods tend to have large execution times which may be longer than the deadlines of other tasks. Once there are more of these tasks than processors, global non-pre-emptive scheduling becomes infeasible.

For the partitioned scheduling algorithms (Figure 11), the weighted schedulability measure is significantly higher than it is for the equivalent global scheduling methods. Here, we again observe that deferred pre-emption (pFPDS) shows the largest improvement over fully pre-emptive scheduling (pFPPS) when the range of task periods is relatively small. This is due to the ability of tasks to tolerate significant blocking in comparison to the execution time of other tasks. With partitioned scheduling, this effect is more pronounced as FNRs can only cause blocking to tasks allocated to the same processor. Hence allocation of tasks with similar execution times (FFMaxC) to the same processor can improve schedulability. As the range of task periods increases, then Decreasing Density becomes by some margin the most effective task allocation heuristic, and the performance of pFPPS using that heuristic tends towards that of pFPDS.

Finally, we observe that with partitioned non-pre-emptive scheduling (pFPNS), the degradation in performance with an increasing range of task periods is less severe than with the equivalent global approach (gFPNS). This is because if there are more than m tasks whose execution time exceeds the smallest deadline, then the entire system is unschedulable with gFPNS, whereas with pFPNS if these tasks are on different processors to those with short deadlines, then the system may still be schedulable.

8. SUMMARY AND CONCLUSIONS

Global fixed priority scheduling with deferred pre-emption (gFPDS), dominates both global fixed priority fully pre-emptive (gFPPS) and global fixed priority non-pre-emptive scheduling (gFPNS). In this paper we provided analysis for a simple model of gFPDS on homogeneous multiprocessors, where each task has a single non-pre-

emptive region at the end of its execution. We showed that an appropriate choice of the length of this region can enhance schedulability.

The main contributions of this paper are as follows:

- Introduction of sufficient schedulability tests for gFPDS.
- Proof that the FNR algorithm [Davis and Bertogna 2012] is compatible with the DA and DA-LC tests for gFPDS, and can be used to obtain the optimal final non-preemptive region lengths for a given priority ordering.
- Proof via a counterexample, that the joint problem of priority and FNR length assignment cannot be solved optimally via a greedy, bottom-up approach using the FNR-PA Algorithm from [Davis and Bertogna 2012].
- An experimental evaluation of the performance benefits of the deferred pre-emption approach for both global and partitioned scheduling. In both cases, (gFPDS v. gFPPS, and pFPDS v. pFPPS) our experiments showed that the use of Final Non-pre-emptive Regions (FNRs) can significantly improve schedulability, particularly when the range of task periods is relatively small.
- While the partitioned approaches were shown to be significantly more effective in our experiments, much of this advantage may be due to pessimism in the underlying schedulability tests for global fixed priority scheduling. Global scheduling should not be discounted as it has significant advantages for open systems.
- We made additional comparisons with the dynamic scheduling algorithm FPZL, these showed that much of the improvement FPZL obtains over gFPPS can be achieved by the simple adoption of FNRs (i.e. gFPDS).

Building on our work on global scheduling with deferred pre-emption, there are two key areas which we aim to explore further.

Firstly, in single processor systems, tasks may execute as a series of non-pre-emptive regions with pre-emption points between them [Bertogna et al. 2011b]. However, with global fixed priority scheduling with eager pre-emption, such an arrangement can potentially be ineffective in the multiprocessor case. This is illustrated in Figure 1 (in Section 4), which shows that there is the potential for every non-pre-emptive region of every lower priority task to interfere with the execution of a higher priority task. This problem is addressed by the lazy pre-emption mechanism of link-based scheduling [Block et al. 2007; Brandenburg 2011]. The existing approach to schedulability analysis for link-based global scheduling [Block et al. 2007; Brandenburg 2011] relies on using a global schedulability test for fully pre-emptive scheduling, with the additional delays due to non-pre-emptive regions accounted for by inflating the worst-case execution time of every task before applying the test [Brandenburg and Anderson 2014]. Further, the amount by which the execution time of each task is inflated is given by the maximum length of any non-pre-emptive region of a lower priority task. In the context of the work reported in this paper, such analysis cannot improve upon the schedulability obtained with global fixed priority pre-emptive scheduling (i.e. with no final non-pre-emptive regions). In general; however, the schedulability of tasks with non-pre-emptable regions under global fixed priority scheduling with eager pre-emption is incomparable to that with lazy pre-emption, as shown by the examples in the appendix. Typically link-based scheduling can be expected to perform better if many tasks have many non-pre-emptive regions of a similar size, whereas eager pre-emption can be expected to perform better if only a few low priority tasks have long final non-pre-emptive regions. An interesting area for future work is the development of schedulability analysis for systems using global fixed priority scheduling with lazy pre-emption, accounting for the improvements in schedulability that can be obtained by virtue of final non-pre-emptive regions. Such analysis would facilitate a performance

comparison between lazy and eager pre-emption in this context. Some preliminary work in this area is reported in [Marinho et al. 2013].

Secondly, our simple model assumes that task execution times are independent of pre-emption and pre-emption and migration costs are negligible; however, in many real-time systems each pre-emption and migration incurs a significant cost, particularly in systems using cache. For large tasksets, allowing arbitrary pre-emption can result in lower priority tasks being pre-empted a large number of times, significantly increasing cache-related pre-emption delays (CRPD) to the detriment of schedulability [Altmeyer at al. 2011; 2012]. The integration of CRPD and schedulability analysis is a key area which we aim to explore further. An indication of the effects of CRPD on the schedulability of tasksets under partitioned FPPS, assuming a separate cache per processor, can be obtained by considering the single processor case [Lunniss et al. 2014]. Deferred pre-emption can typically be expected to reduce the effect of CRPD in such systems, since it reduces the number of pre-emptions. Until effective analysis is developed for global FPPS incorporating CRPD, it is not possible to say whether these effects will be larger or smaller for global scheduling than partitioned. Certainly there is scope for large pre-emption and migration delays in globally scheduled systems; however, the extent of these effects clearly depends on the cache configuration: shared between processors and tasks, shared between processors but partitioned between tasks or groups of tasks, or a separate cache per processor.

ELECTRONIC APPENDIX

The electronic appendix for this article can be accessed in the ACM Digital Library.

REFERENCES

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

S. Altmeyer, R.I. Davis, C. Maiza. 2012. “Improved cache related pre-emption delay aware response time analysis for fixed priority pre-emptive systems” . Real-Time Systems, 48 (5), pp. 499-526.

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

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

T.P. Baker. 2003. “Multiprocessor EDF and deadline monotonic schedulability analysis”. In proceedings. Real-Time Systems Symposium (RTSS), pp. 120–129.

S.K. Baruah, A. Burns. 2006.“Sustainable Scheduling Analysis”. In proceedings Real-Time Systems Symposium (RTSS), pp. 159-168.

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

S.K. Baruah. 2007. “Techniques for Multiprocessor Global Schedulability Analysis”. In proceedings Real-Time Systems Symposium (RTSS), pp. 119-128.

.S.K. Baruah, N. Fisher. 2008. “Global Fixed-Priority Scheduling of Arbitrary-Deadline Sporadic Task Systems” In proceedings International Conference on Distributed Computing and Networking, pp. 215-226.

A. Bastoni, B. Brandenburg, and J. Anderson. 2010. "Cache-Related Preemption and Migration Delays: Empirical Approximation and Impact on Schedulability" In Proceedings of OSPERT, pp. 33-44.

M. Bertogna, M. Cirinei, G. Lipari. 2005. “New schedulability tests for real-time task sets scheduled by deadline monotonic on multiprocessors”. In proceedings International Conf. on Principles of Distributed Systems, pp. 306-321.

M. Bertogna, M. Cirinei. 2007. “Response Time Analysis for global scheduled symmetric multiprocessor platforms”. In proceedings Real-Time Systems Symposium (RTSS), pp. 149-158.

M. Bertogna, M. Cirinei, G. Lipari. 2009. “Schedulability analysis of global scheduling algorithms on multiprocessor platforms”. IEEE Transactions on parallel and distributed systems, 20(4): 553-566..

M. Bertogna, G. Buttazzo, M. Marinoni, G. Yao, Francesco Esposito, Marco Caccamo. 2010. "Preemption points placement for sporadic task sets", In Proceedings Euromicro Conference on Real-Time Systems (ECRTS).

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

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

A. Block, H. Leontyev, B. Brandenburg, J.H. Anderson. 2007. "A Flexible Real-Time Locking Protocol for Multiprocessors". In Proceedings of Real-Time Computing Systems and Applications (RTCSA) , pp 47-56.

M. Blum, R..W. Floyd, V. Pratt, R.L. Rivest, R.E. Tarjan. 1973. "Time bounds for selection". Journal of Computer and System Sciences 7, 4, 448–461.

B. B. Brandenburg, J. Anderson. 2014. "A Clarification of Link-based Global Scheduling", Max Plank Institute for Software Systems, Technical Report MPI-SWS-2014-007. Available from http://www.mpi-sws.org/cont/tr/2014-007.pdf.

B. B. Brandenburg. 2011. "Scheduling and Locking in Multiprocessor Real-Time Operating Systems", PhD Thesis, The University of North Carolina at Chapel Hill.

R. Bril, J. Lukkien, and W. Verhaegh. 2009. Worst-case response time analysis of real-time tasks under fixed-priority scheduling with deferred preemption. Real-Time Systems, 42(1-3):63–119.

A. Burns. 1994. "Preemptive priority based scheduling: An appropriate engineering approach". S. Son, editor, Advances in Real-Time Systems, pp. 225–248.

A. Burns, S.K. Baruah. 2008. "Sustainability in real-time scheduling". Journal of Computing Science and Engineering 2 (1), pp 74-97.

G.C. Buttazzo, M. Bertogna, G. Yao. 2013. "Limited Preemptive Scheduling for Real-Time Systems: A Survey". IEEE Transactions on Industrial Informatics, 9(1) pp. 3-15.

R.I. Davis, A. Burns. 2009. "Priority Assignment for Global Fixed Priority Pre-emptive Scheduling in Multiprocessor Real-Time Systems". In proceedings Real-Time Systems Symposium (RTSS), pp. 398-409.

R.I. Davis, T. Rothvoß, S.K. Baruah, A. Burns. 2009. "Exact Quantification of the Sub-optimality of Uniprocessor Fixed Priority Pre-emptive Scheduling". Real-Time Systems, 43, 3, pp. 211-258.

R.I. Davis, A. Burns, 2011a. "Improved Priority Assignment for Global Fixed Priority Pre-emptive Scheduling in Multiprocessor Real-Time Systems". Real-Time Systems 47 (1) pp1-40.

R.I. Davis, A. Burns. 2011b. "FPZL Schedulability Analysis", In proceedings Real-Time Applications and embedded Technology Symposium (RTAS), pp. 245-256.

R.I. Davis and S. Kato. 2012. "FPSL, FPCL and FPZL schedulability analysis." Real-Time Systems, 48 (12), pp 750-788.

R.I. Davis, A. Burns. 2011c. "A Survey of Hard Real-Time Scheduling for Multiprocessor Systems", ACM Computing Surveys, 43, 4, Article 35 44 pages.

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

R.I.Davis, A. Burns, R.J. Bril, and J.J. Lukkien. 2007. "Controller Area Network (CAN) Schedulability Analysis: Refuted, Revisited and Revised". *Real-Time Systems*, Volume 35, Number 3, pp. 239-272.

R.I. Davis, A. Burns, J. Marinho, V. Nelis, S.M. Petters, M. Bertogna. 2013. "Global Fixed Priority Scheduling with Deferred Pre-emption", In proceedings International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA).

P. Emberson, R. Stafford, R.I. Davis. 2010. "Techniques For The Synthesis Of Multiprocessor Tasksets". In proceedings 1st International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS) , pp. 6-11.

N. Fisher, S.K. Baruah. 2006. "Global Static-Priority Scheduling of Sporadic Task Systems on Multiprocessor Platforms." In procedings. IASTED International Conference on Parallel and Distributed Computing and Systems.

N. Fisher, S.K. Baruah, T.P. Baker. 2006. "The partitioned scheduling of sporadic tasks according to static priorities". In proceedings of the EuroMicro Conference on Real-Time Systems (ECRTS), pp. 118–127.

M. Garey and D. Johnson. 1979. Computers and Intractability: a Guide to the Theory of NP-Completeness. W. H. Freeman and company, NY.

N. Guan, W. Yi, Q. Deng, Z. Gu, G. Yu. 2011. "Schedulability analysis for non-preemptive fixed-priority multiprocessor scheduling". Journal of Systems Architecture - Embedded Systems Design 57(5), pp. 536-546.

N. Guan, M. Stigge, W.Yi, G. Yu. 2009. "New Response Time Bounds for Fixed Priority Multiprocessor Scheduling". In proceedings of the Real-Time Systems Symposium (RTSS), pp. 388-397.

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

W. Lunniss, S. Altmeyer, R.I. Davis. 2014. "A Comparison between Fixed Priority and EDF Scheduling accounting for Cache Related Pre-emption Delays". Leibniz Transactions on Embedded Systems (LITES), Volume 1, Number 1. DOI: http://dx.doi.org/10.4230/LITES-v001-i001-a001.

J. Marinho, V. Nelis, S.M. Petters, M. Bertogna, R.I.Davis. 2013. "Limited Pre-emptive Global Fixed Task Priority". In Proceedings Real-Time Systems Symposium (RTSS), pp. 182-191.

J. Marinho,V. Nélis,S. M. Petters, I. Puaut. 2012. "Preemption Delay Analysis for Floating Non-Preemptive Region Scheduling", In proceedings of DATE.

D.I. Oh, T.P. Baker. 1998. "Utilization bounds for N-processor rate monotone scheduling with stable processor assignment". Real Time Systems, 15(2):183–193.

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

Received September 2013; revised November 2014; accepted January 2015

Online Appendix to: Global and Partitioned Multiprocessor Fixed Priority Scheduling with Deferred Pre-emption

ROBERT I. DAVIS, University of York
ALAN BURNS, University of York
JOSE MARINHO, CISTER/INESC-TEC, ISEP
VINCENT NELIS, CISTER/INESC-TEC, ISEP
STEFAN M. PETTERS, CISTER/INESC-TEC, ISEP
MARKO BERTOGNA, University of Modena

A. SPECIAL CASE OF A FULLY NON-PRE-EMPTIVE TASK IN A gFPDS SYSTEM

We note that while our schedulability tests for gFPDS dominate the equivalent tests for gFPPS, they do not dominate the equivalent tests for gFPNS which include blocking from at most m lower priority tasks [Guan et al. 2011]. We can however apply specific schedulability tests for gFPDS for the special case of a task τ_k which is fully non-pre-emptive, as set out below.

In the case of gFPDS scheduling where a task τ_k is fully non-pre-emptive, i.e. when $F_k = C_k$, $D_k^* = D_k - (C_k - 1)$, and $C_k^* = 1$, then more precise analysis is possible. This analysis is based on the approach of [Guan et al. 2011] for gFPNS (where *all* tasks are fully non-pre-emptive).

Following the approach of [Guan et al. 2011], we can limit the number of carry-in jobs considered. We again use the concept of a *problem window*, which relates to a job of task τ_k missing its deadline. Recall that J_k is the first job of τ_k that misses its deadline, with r_k being the release time of J_k and d_k its absolute deadline. Further, let $l_k = d_k - (C_k - 1)$ be the latest time by which the job must have started its non-pre-emptable execution in order to complete by its deadline (see Figure 12).

We again use $\Psi(t,k)$ to denote the set of tasks with higher priority than τ_k (i.e. $hp(k)$) that have active jobs (i.e. jobs that have been released but not yet completed) at time t. Further t_0 is defined, as in Section 4.1, to be the earliest time before r_k such that $\forall t \in [t_0, r_k)$ at least one of the following holds:

(i) $|\Psi(t,k)| = m$ and all of the tasks in $\Psi(t,k)$ execute in the interval $[t, t+1)$.

(ii) There are some tasks in $\Psi(t,k)$ that do not execute in the interval $[t, t+1)$.

If there is no such t_0, then $t_0 = r_k$. The start of the problem window is defined as t_0. The problem window has a fixed length $L = D_k - (C_k - 1)$ and so ends at $t_f = t_0 + L$. From the definition of t_0, we note that no job of a task with priority k or lower can start to execute during the interval $[t_0, r_k)$.

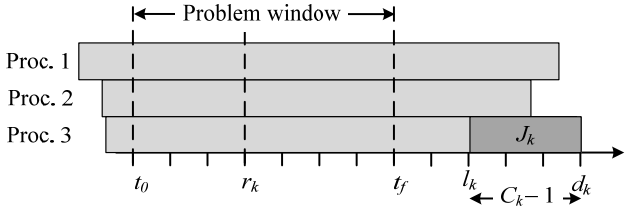

Figure 12: Problem window for a fully non-pre-emptive task.

We observe that Lemma 1 and Lemma 2 (in Section 4.1) hold. Thus at most m-1 tasks with a higher priority than k can have a carry-in job, and the maximum interference from all previous jobs (prior to J_k) of task τ_k in the problem window is $(F_k - 1) = (C_k - 1)$.

LEMMA A.1: At most m tasks in total can have a carry-in job that interferes with J_k in the problem window.

PROOF: (Follows the logic of the proof of Lemma 5.2 in [Guan et al. 2011]). At time $t_0 - 1$ then given how t_0 is defined, it follows that $|\Psi(t_0 - 1, k)| \neq m$ and *all* of the tasks in $\Psi(t_0 - 1, k)$ execute in the interval $[t_0 - 1, t_0)$. A task in $hp(k)$ only has a carry-in job if it is in $\Psi(t_0 - 1, k)$, hence $|\Psi(t_0 - 1, k)|$ tasks in $hp(k)$ have carry-in jobs. A task with priority k or lower can only have a carry-in job that interferes with J_k if it is executing a FNR in the interval $[t_0 - 1, t_0)$. Since at most m tasks are executing in the interval $[t_0 - 1, t_0)$, and $|\Psi(t_0 - 1, k)|$ of them are in $hp(k)$, the number of tasks in $lep(k)$ with carry-in jobs is $m - |\Psi(t_0 - 1, k)|$. Hence the maximum number of tasks with carry in jobs that interfere with J_k is m □

We note that as task τ_k is fully non-pre-emptive, a lower priority task τ_j can only interfere with the execution of J_k if it has a carry-in job that is executing a FNR at the start of the problem window. Otherwise J_k will start executing in preference to τ_j, and once J_k starts executing it runs to completion. Hence the maximum interference from a lower priority task τ_j is zero if it does not have a carry-in job and $F_j - 1$ if it has a carry-in job. So, for the virtual task τ_{jv} representing τ_j, we have $I_{jv}^{D}(L, C) = I_{jv}^{R}(L, C) = F_j - 1$, and $I_{jv}^{NC}(L, C) = 0$ and hence $I_{jv}^{DIFF-D}(L, C) = I_{jv}^{DIFF-R}(L, C) = F_j - 1$.

We now compile a set of sufficient schedulability tests for gFPNS.

In the simple DA test below, the effect of push-through blocking from the FNR of the previous job of task τ_k is factored into the interference term for higher priority tasks, as was the case with gFPDS (see Section 4.1). Hence we only include the additional interference term from the FNRs of tasks with strictly lower priority (and we do not limit the number of them).

DA test: *for a fully non-pre-emptive task τ_k under gFPDS:*

$$D_k^* \geq C_k^* + \left\lfloor \frac{1}{m} \left(\sum_{\forall i \in hp(k)} I_i^D(D_k^*, C_k^*) + \sum_{\forall j \in lp(k)} (F_j - 1) \right) \right\rfloor \tag{A.1}$$

The more sophisticated DA-LC test makes use of Lemma 1, Lemma 2, and Lemma A.1 to limit the amount of carry-in interference.

DA-LC test: *for a fully non-pre-emptive task τ_k under gFPDS:*

$$D_k^* \geq C_k^* + \left\lfloor \frac{1}{m} \left(\sum_{\forall i \in hp(k)} I_i^{NC}(D_k^*, C_k^*) + \sum_{i \in MDB(k)} I_i^{DIFF-D}(D_k^*, C_k^*) \right) \right\rfloor \tag{A.2}$$

where $MDB(k)$ is the subset of the m tasks with the largest values of $I_i^{DIFF-D}(D_k^*, C_k^*)$ from the set of tasks $hp(k) \cup lepv(k)$ provided at least one of those tasks is from $lepv(k)$, otherwise $MDB(k)$ equates to the subset of at most m-1 tasks with the largest values of $I_i^{DIFF-D}(D_k^*, C_k^*)$ given by (8), from the set of tasks $hp(k)$, and the single virtual task from $lepv(k)$ that has the largest value of $I_i^{DIFF-D}(D_k^*, C_k^*)$. Note $lepv(k)$ is the set of virtual tasks representing the FNRs of tasks with priority k or lower.

RTA test: *The upper bound response time R_k^S for the start (first unit of execution) of a fully non-pre-emptive task τ_k under gFPDS, may be computed via the fixed point iteration given by (A.3) within Algorithm 1. The task is schedulable if $R_k^S \leq D_k^*$, where D_k^* is the task's effective deadline $D_k^* = D_k - (C_k - 1)$.*

$$R_k^S \leftarrow C_k^* + \left\lfloor \frac{1}{m} \left(\sum_{\forall i \in hp(k)} I_i^R(R_k^S, C_k^*) + \sum_{\forall j \in lp(k)} (F_j - 1) \right) \right\rfloor \tag{A.3}$$

If the task is schedulable, then an upper bound on its worst-case response time is given by $R_k^{UB} = R_k^S + (C_k - 1)$.

RTA-LC test: *The upper bound response time R_k^S for the start (first unit of execution) of a fully non-pre-emptive task τ_k under gFPDS, may be computed via the*

fixed point iteration given by (A.4) within Algorithm 1. The task is schedulable if $R_k^S \le D_k^*$*, where* D_k^* is *the task's effective deadline* $D_k^* = D_k - (C_k - 1)$.

$$R_k^S \leftarrow C_k^* + \left\lfloor \frac{1}{m} \left(\sum_{\forall i \in hp(k)} I_i^{NC}(R_k^S, C_k^*) + \sum_{i \in MRB(k)} I_i^{DIFF-R}(R_k^S, C_k^*) \right) \right\rfloor \quad \text{(A.4)}$$

where $MRB(k)$ is the subset of the m tasks with the largest values of $I_i^{DIFF-R}(R_k^S, C_k^*)$ from the set of tasks $hp(k) \cup lepv(k)$ provided at least one of those tasks is from $lepv(k)$, otherwise $MDB(k)$ equates to the subset of at most m-1 tasks with the largest values of $I_i^{DIFF-R}(R_k^S, C_k^*)$ given by (14), from the set of tasks $hp(k)$, and the single virtual task from $lepv(k)$ that has the largest value of $I_i^{DIFF-R}(R_k^S, C_k^*)$. If the task is schedulable, then an upper bound on its worst-case response time is given by $R_k^{UB} = R_k^S + (C_k - 1)$.

We note that the RTA and RTA-LC tests given by (A.3) and (A.4) do not depend on the upper bound response times of lower priority tasks, and so the iteration of Algorithm 1 is unnecessary if all tasks are fully non-pre-emptive. In that case, upper bound response times may be evaluated highest priority first.

We observe that the schedulability tests given in this section for the special case of a fully non-pre-emptive task dominate the equivalent tests for the general case of deferred pre-emption with $F_k = C_k$ given in section 4. This means that the DA test retains it monotonic behaviour with respect to increasing values of F_k if in the special case where $F_k = C_k$ we use the specific test given by (A.1) instead of the more general one given by (4).

B. LINK-BASED SCHEDULING

Link-based scheduling introduced by [Block et al. 2007], and further described by [Brandenburg 2011] in section 3.3.3 of his thesis, uses a lazy pre-emption mechanism with the aim of avoiding issues of repeated blocking which can occur when tasks execute non-pre-emptive regions under global multiprocessor scheduling. Note in the following description we refer to jobs since link-based scheduling can be applied to fixed job priority scheduling algorithms, such as global EDF, as well as fixed task priority algorithms such as global fixed priority scheduling.

Link-based scheduling uses the concept of a link between a job and a processor. At any given time, the m highest priority ready jobs are linked to the m processors. The idea is that a link records where a job would be scheduled if all of the jobs were pre-emptable all of the time [Brandenburg 2011]. When a job J_A is released, then if there is a processor that is idle, it is linked to that processor. Otherwise, if J_A has a higher priority than the lowest priority linked job J_B, then J_B is unlinked and J_A is linked to the processor that J_B was previously linked to. A job that is linked to a processor (e.g. J_A) pre-empts the previous job (e.g. J_B) executing on that processor as soon as the latter becomes pre-emptable. Further, when a job completes on a processor p then the job (if any) linked to that processor executes, otherwise the highest priority unlinked job is linked to processor p and executed on that processor. (Full details of the link-based scheduling mechanism are given in [Block et al. 2007] and section 3.3.3 of [Brandenburg 2011]).

With link-based scheduling pre-emptions are lazy in the sense that if the lowest priority running job J_B is non-pre-emptable and executing on processor p, then a high priority job J_A will be linked to that processor and so will not pre-empt any other medium priority job (e.g. J_C) executing on another processor even though J_C is pre-emptable and has a lower priority than J_A.

B.1 Incomparability between lazy and eager pre-emption

Depending on the task parameters and non-pre-emptive region lengths, the lazy pre-emption mechanism of link-based scheduling may improve or diminish schedulability compared to global fixed priority scheduling with eager pre-emption. An example of such incomparability is given below.

First, consider a two processor system with three tasks with the parameters given in Table IV. Task τ_A has the highest priority and task τ_C the lowest. Tasks τ_A and τ_B are fully pre-emptable, whereas task τ_C is fully non-pre-emptable.

Table IV: Task parameters

Task	Execution time	Period	Deadline
τ_A	5	100	10
τ_B	8	100	100
τ_C	10	100	100

Under link based scheduling, this system is unschedulable as shown in Figure 13(a), since if tasks τ_B and τ_C are released at the same time and start to execute neither may be pre-empted by task τ_A which is linked to processor 1 and inevitably misses its deadline. This system is however easily scheduled with eager pre-emption, since in that case task τ_A can immediately pre-empt task τ_B and so suffers no blocking (see Figure 13(b)).

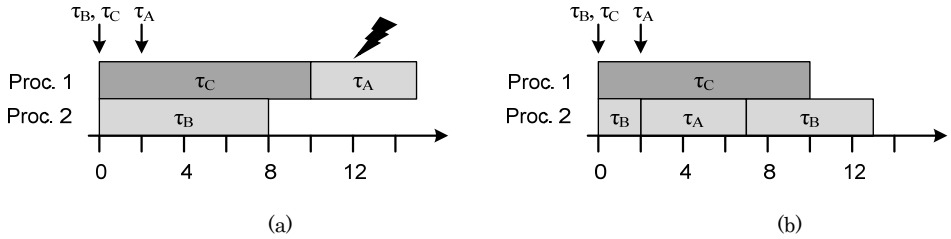

Figure 13: Schedule with (a) lazy pre-emption (b) eager pre-emption.

Second, consider a two processor system with three tasks with the parameters given in Table V below. Again, task τ_A has the highest priority and task τ_C the lowest. This time, task τ_C has three non-pre-emptable regions, each of length 2, as shown in darker grey in Figure 14. Tasks τ_A and τ_B are fully pre-emptable.

Table V: Task parameters

Task	Execution time	Period	Deadline
τ_A	2	6	6
τ_B	10	100	14
τ_C	14	100	100

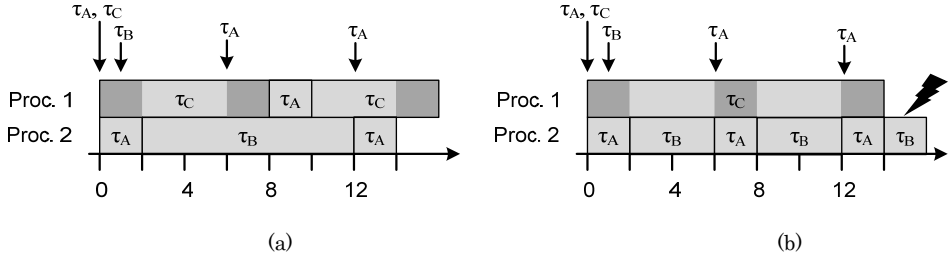

Figure 14: Schedule with (a) lazy pre-emption (b) eager pre-emption.

With eager pre-emption (as shown in Figure 14(b)), the system is unschedulable. This is because each time task τ_A is released (i.e. at times 0, 6 and 12), task τ_C is

executing a non-pre-emptable region, and so task τ_A pre-empts task τ_B which subsequently misses its deadline. Effectively task τ_B suffers blocking due to all three non-pre-emptive regions of task τ_C. In this particular example, this is avoided when the lazy pre-emption mechanism of link-based scheduling is employed, as shown in Figure 14(a). Here, when task τ_A is released at time 6, it suffers blocking of 2 time units and does not pre-empt task τ_B. This allows task τ_B to meet its deadline. This example illustrates the trade-off involved in lazy pre-emption, the response time of task τ_A is increased, while the response time of task τ_B is decreased. Depending on the deadlines of the two tasks this may either be advantageous as in this example or detrimental as in the previous example.

Together these two examples show that the schedulability of systems with non-pre-emptive regions under global fixed priority scheduling with eager and lazy pre-emption mechanisms are incomparable. Depending on the specific task parameters, either lazy or eager pre-emption may lead to better schedulability.