Technical Report

Preemption-light multiprocessor scheduling of sporadic tasks with high utilisation bound

Konstantinos Bletsas

Björn Andersson

HURRAY-TR-110105

Version:

Date: 01-14-2011

Preemption-light multiprocessor scheduling of sporadic tasks with high utilisation bound

Konstantinos Bletsas, Björn Andersson

IPP-HURRAY!

Polytechnic Institute of Porto (ISEP-IPP)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8340509

E-mail:

http://www.hurray.isep.ipp.pt

Abstract

Known algorithms capable of scheduling implicit-deadline sporadic tasks over identical processors at up to 100% utilisation invariably involve numerous preemptions and migrations. To the challenge of devising ascheduling scheme with as few preemptions and migrations as possible, for a given guaranteed utilisation bound, we respond with the algorithm NPS-F. It is configurable with a parameter, trading off guaranteed schedulable utilisation (up to 100%) vs preemptions. For any possible configuration, NPS-F introduces fewer preemptions than any other known algorithm matching its utilisation bound.

A clustered variant of the algorithm, for systems made of multicore chips, eliminates (costly) off-chip task migrations, by dividing processors into disjoint clusters, formed by cores on the same chip (with the cluster size being a parameter). Clusters are independently scheduled (each, using non-clustered NPS-F). The utilisation bound is only moderately affected.

We also formulate an important extension (applicable to both clustered and non-clustered NPS-F) which optimises the supply of processing time to executing tasks and makes it more granular. This reduces processing capacity requirements for schedulability without increasing preemptions.

Noname manuscript No.
(will be inserted by the editor)

Preemption-light multiprocessor scheduling of sporadic tasks with high utilisation bound

Konstantinos Bletsas · Björn Andersson

Received: date / Accepted: date

Abstract Known algorithms capable of scheduling implicit-deadline sporadic tasks over identical processors at up to 100% utilisation invariably involve numerous preemptions and migrations. To the challenge of devising a scheduling scheme with as few preemptions and migrations as possible, for a given guaranteed utilisation bound, we respond with the algorithm NPS-F. It is configurable with a parameter, trading off guaranteed schedulable utilisation (up to 100%) vs preemptions. For any possible configuration, NPS-F introduces fewer preemptions than any other known algorithm matching its utilisation bound.

A clustered variant of the algorithm, for systems made of multicore chips, eliminates (costly) off-chip task migrations, by dividing processors into disjoint clusters, formed by cores on the same chip (with the cluster size being a parameter). Clusters are independently scheduled (each, using non-clustered NPS-F). The utilisation bound is only moderately affected.

We also formulate an important extension (applicable to both clustered and non-clustered NPS-F) which optimises the supply of processing time to executing tasks and makes it more granular. This reduces processing capacity requirements for schedulability without increasing preemptions.

Keywords multiprocessor scheduling · utilisation bound · multicore · preemptions · migrations

A preliminary version of this paper can be found in the Proceedings of the 30th IEEE Real-Time Systems Symposium, 2009, pp 447–456.

K. Bletsas
CISTER/ISEP Research Centre, Polytechnic Institute of Porto, Portugal
Tel.: +351-228340529
Fax: +351-228340509
E-mail: ksbs@isep.ipp.pt (official)
E-mail: mpletsas@gmail.com (for correspondence during review process)

B. Andersson
CISTER/ISEP Research Centre, Polytechnic Institute of Porto, Portugal
E-mail: bandersson@dei.isep.ipp.pt

1 Introduction

Consider the problem of preemptively scheduling n sporadic tasks (τ_1 to τ_n) on m identical processors (P_1 to P_m). A task τ_i generates a (potentially infinite) sequence of jobs, occurring at least T_i time units apart. A job by τ_i requires up to C_i time units of execution over the next T_i time units after its arrival (with T_i, C_i being real numbers and $0 \leq C_i \leq T_i$). This model is termed *sporadic*; it is a generalisation of the classic periodic model. We assume that tasks share no resources (other than the processors themselves). A processor executes at most one job at a time and no job may execute on multiple processors simultaneously. The utilisation of the task set is defined as $U_\tau = \frac{1}{m} \cdot \sum_{i=1}^{n} \frac{C_i}{T_i}$. The utilisation bound UB of a scheduling algorithm is a threshold such that all task sets with $U_\tau \leq \text{UB}$ meet their deadlines under said algorithm.

Many multiprocessor scheduling algorithms can be categorised as either *partitioned* or *global*. Under global scheduling, a single dispatch queue is shared by all processors and at any moment, the m highest-priority runnable tasks get to execute on the m processors. Under partitioning, each task may only execute on a specific processor and not migrate. Preemptions are limited and the multiprocessor scheduling problem is reduced to many uniprocessor scheduling problems (which allows reuse of many results from uniprocessor scheduling). Yet no partitioned algorithm can have a utilisation bound above 50% (Carpenter et al, 2004). Conversely, the *pfair* family of global scheduling algorithms offers utilisation bounds of 100% (Baruah et al, 1996; Anderson and Srinivasan, 2004) but at the cost of numerous preemptions (Devi and Anderson, 2005).

Hybrid approaches aim for combination of strengths: EDF-fm (Anderson et al, 2005) schedules soft, not hard, real-time tasks at up to 100% system utilisation with limited tardiness. Ehd2-SIP and EDDP (Kato and Yamasaki, 2007, 2008), with utilisation bounds of 50% and 65%, typically generate few preemptions, although no respective upper bound is known. Under EKG-sporadic (Andersson and Bletsas, 2008) most tasks utilise a single processor but at most $m - 1$ utilise two – but never simultaneously. This algorithm is configurable for utilisation bounds up to 100% at the cost of increased preemptions.

Among other hybrid approaches, EDF-SS (Andersson et al, 2008) is related to EKG-sporadic, but extends to arbitrary-deadline tasks. Due to different bin-packing and task-splitting, for implicit-deadline tasks, it performs better than EKG-sporadic on average – but has no proven utilisation bound. By comparison, EDF-WM (Kato et al, 2009) involves much fewer preemptions for similar performance, and is proven to dominate partitioning – but does not have a proven utilisation bound above 50%. Finally, DM-PM (Kato and Yamasaki, 2009), another scheme with limited preemptions, is based on fixed-priority scheduling (unlike the other algorithms, which are based on EDF). Again, the utilisation bound is 50%.

Before NPS-F (which was first presented in the conference version of this paper (Bletsas and Andersson, 2009b)), of all schemes with utilisation bounds

above 50%, the most "preemption-light" (in terms of a proven upper bound on preemptions) was Notional Processor Scheduling (NPS) (Bletsas and Andersson, 2009a). Yet its utilisation bound is just $66.\bar{6}\%$. Note that although EDZL (Cho et al, 2002) has even fewer preemptions, it is yet unproven whether its utilisation bound exceeds 50% (Chao et al, 2008).

NPS-F, is configurable for utilisation bounds from 75% up to 100% (traded off with preemptions). NPS-F has better upper bounds on preemptions than any known algorithm matching it in terms of utilisation bound. We also introduce a clustered variant of NPS-F (for systems made of multicore chips) with somewhat lower utilisation bound but which eliminates the costliest (in terms of overhead) type of migrations. These are the migrations between cores on different chips, which cause cache misses necessitating main memory I/O and severely hurting performance (Fedorova et al, 2005; Anderson et al, 2006).

The scheduling potential of NPS-F can be noticeably improved if the offsets between the time windows of execution on different processors, for migrating tasks, are optimised. In this article, we introduce a sophisticated approach to that, termed Ω-optimisation. This optimisation reduces the processing capacity requirements for scheduling a given set of tasks.

NPS-F stands for "Notional Processor Scheduling – Fractional capacity". It is related to NPS (Bletsas and Andersson, 2009a) though the reader need not be familiar. The relation is discussed later.

In this paper, Section 2 discusses concepts prerequisite to understanding the approach. Section 3 introduces NPS-F and quantifies its performance in terms of schedulable utilisation and preemptions. Section 4 does the same for the clustered variant. Section 5 introduces the Ω-optimisation, which reduces processing capacity requirements for schedulability. Section 6 experimentally evaluates the performance of the algorithm, for scheduling task sets with utilisation above its utilisation bound. Section 7 discusses principles for the integration of shared resource management to the scheduling scheme. Section 8 concludes.

2 Useful concepts

2.1 On periodic reserves

We assume a sporadic task model (Mok, 1983; Leung and Whitehead, 1982) (as do also virtually all of the previously cited hybrid multiprocessor algorithms). This task model is tremendously relevant for its usefulness in modelling real-time systems (Fisher et al, 2010, p. 27). We also assume implicit deadlines (i.e. the deadline of task τ_i is T_i).

A *periodic reserve* is a kind of server, for scheduling one or more sporadic tasks. The concept originates with our work in (Andersson and Bletsas, 2008). It is a fixed-length time window, available periodically, every S time units (an interval termed the "timeslot length"). Time within the reserve is *exclusively* for the execution of tasks served by it (e.g. under EDF policy). Conversely,

tasks served by the reserve cannot execute outside its bounds. Figure 1(a) provides an example of multiple tasks scheduled within a periodic reserve, under EDF.

Suppose that $S \leq \frac{\text{TMIN}}{\delta}$, where TMIN is the shortest of the interarrival times of all tasks served by the reserve and δ is a positive integer. It then holds (see Theorem 5 in the Appendix) that implicit-deadline tasks of cumulative utilisation U are always schedulable under EDF by a reserve of length $\text{inflate}(U) \cdot S$, with $\text{inflate}(U)$ given by:

$$\text{inflate}(U) = \frac{(\delta + 1) \cdot U}{U + \delta} \tag{1}$$

The inverse function,

$$\text{deflate}(U) \stackrel{\text{def}}{=} \text{inflate}^{-1}(U) = \frac{\delta \cdot U}{(\delta + 1) - U} \tag{2}$$

represents the maximum cumulative task utilisation that a periodic reserve of size $U \cdot S$ may accommodate (with $S = \frac{\text{TMIN}}{\delta}$). The quantity

$$\alpha(U) \stackrel{\text{def}}{=} \text{inflate}(U) - U = \frac{U \cdot (1 - U)}{U + \delta} \tag{3}$$

is termed *inflation*. It expresses, by which amount the processor capacity allocated for a reserve should exceed the cumulative utilisation of the respective workload served, so as to ensure schedulability of all tasks served by the reserve, even under the most unfavorable arrival phasings, relative to the start/end of a reserve. Figure 2 plots the functions inflate, deflate and α for different values of δ. The higher the value of δ, the lesser the inflation.

2.2 Utilising multiple adjacent reserves

Typically, each reserve is implemented on a particular processor – see Figure 1(a). However, multiple temporally adjacent, "staggered" reserves on different processors, all serving the same tasks, act as a "distributed reserve". In terms of processing capacity for tasks served (ignoring dispatching/migration overheads), such a distributed reserve is equivalent to a conventional reserve of the aggregate length implemented on a single processor (see illustration in Figure 1(b)). In (Bletsas and Andersson, 2009a), when such a distributed reserve has 100% of the processing capacity of a processor, it is dubbed *notional processor*. By extension, if this capacity is below 100%, it is termed *fractional-capacity* notional processor. A notional processor is formally represented as

$$\big((a_0, a_1, ..., a_z), (h_0, h_1, ..., h_{z-1}), \omega\big)$$

with $0 = a_0 < a_1 < \ldots < a_z \leq 1$, $h_u \in \{1, 2, \ldots, m\}$, $\forall u \in \{0, 1, \ldots, z-1\}$ and $0 \leq \omega < 1$. The a variables correspond to time offsets within a timeslot (expressed as a fraction of S) – any two successive a variables define the start

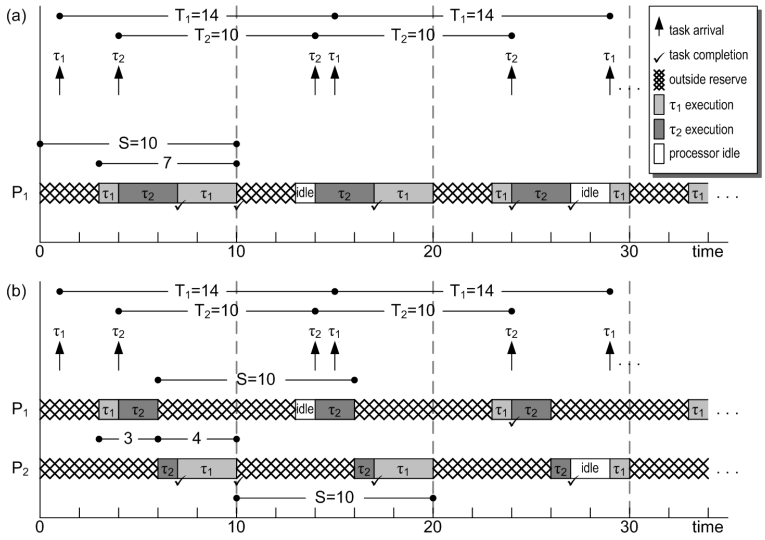

Fig. 1 Scheduling multiple tasks within (a) a single periodic reserve (b) temporally adjacent reserves on different processors. Observe that in example (b), wherein the two reserves summed match in duration the single reserve in the first example (with identical arrivals and timeslot length), task execution intervals are identical to those in case (a).

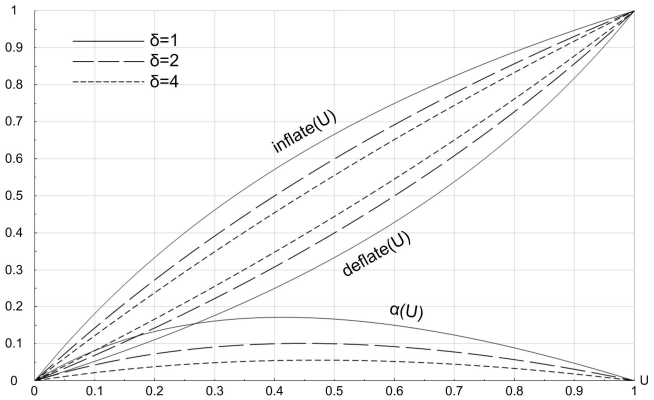

Fig. 2 Functions inflate, deflate and α, for different values of parameter δ.

and end of a reserve. The h variables are indexes of physical processors (where the notional processor is mapped betweeen any two a-offsets). The variable ω is a phase offset (expressed as a fraction of S) for the start of the first reserve, relative to time $t = 0$. The integer z is simply the number of reserves that the notional processor uses.

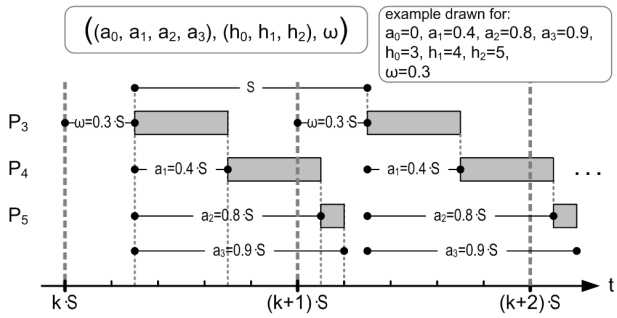

Fig. 3 An example demonstating how the formal definition of a notional processor translates its mapping onto physical processors

The semantics are that on time instant t, the notional processor in consideration is mapped to processor P_{h_r}, r being the integer for which:

$$a_r \cdot S \leq (t - \omega \cdot S) \text{ modulo } S < a_{r+1} \cdot S \tag{4}$$

If $a_z = 1$, the notional processor is *full-capacity* (i.e. always mapped to some physical processor, on every instant); else, it is of fractional capacity deflate(a_z). The example of Figure 3 demonstrates how the formal definition of a notional processor specifies its mapping onto physical processors at runtime.

There is one run queue per notional processor and one dispatcher per physical processor.

3 The algorithm NPS-F

Consider the assignment of n tasks (in no particular order) over infinite unit-capacity *bins* (b_1, b_2, ...) using *First-Fit* bin-packing. (First-Fit assigns tasks one by one to the lowest-indexed bin possible, subject to previous assignments). Post-assignment, as tasks are finite, only some m'' (finite) bins (b_1 to $b_{m''}$) have been assigned tasks. Let U_p denote the cumulative utilisation of tasks assigned to b_p.

Our approach schedules the tasks on each bin b_p on a corresponding notional processor $\tilde{P}_p$ of (fractional) capacity inflate(U_p). In turn, all m'' notional processors are implemented upon the m physical processors (P_1 to P_m).

In an example, Figure 4(a) depicts bin utilisations after bin-packing and Figure 4(b) shows the processing capacity requirements for corresponding notional processors. Note that, to schedule tasks of cumulative utilisation U_p on a notional processor, the latter consumes processing capacity inflate(U_p) $\geq U_p$.

We have come up with two alternative approaches (equivalent, in terms of scheduling potential) for mapping notional processors to physical ones.

Under *flat* mapping (Figure 4(c)), each notional processor (and thus, each task) migrates between at most two physical processors. This is achieved by

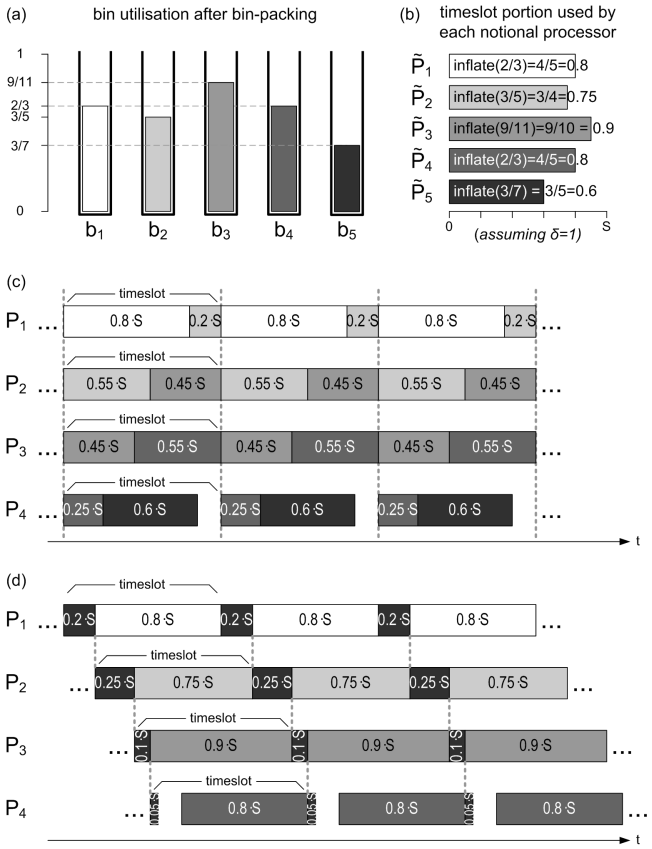

Fig. 4 Alternative approaches to mapping notional to physical processors.

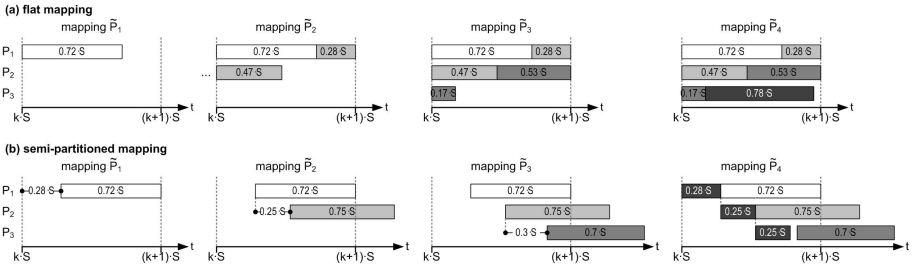

Fig. 5 "Walk through" of how notional processors are mapped, one by one, under (a) the flat and (b) the semi-partitioned approach. In the example, 4 notional processors ($\tilde{P}_1$ to $\tilde{P}_4$, respectively consuming 0.72, 0.75, 0.7 and 0.78 times the processing capacity of a physical processor) are mapped to 3 physical processors (P_1 to P_3). The patterns drawn are repeating.

"filling" physical processors one by one; to accommodate a notional processor, we use one reserve formed out of the remaining capacity of the current physical processor and, if necessary, another reserve (appropriately sized) on the next higher-indexed processor.

Under *semi-partitioned* mapping (Figure 4(d)), notional processors indexed 1 to m are implemented each on a single reserve, on a different notional processor. Leftover processing capacity is then used for the creation of reserves where the remaining notional processors are mapped. Therefore, m notional processors (and the associated tasks) always stay on one physical processor, but the rest may need to migrate between many.

Figure 5 illustrates, via a step-by-step example, how notional processors are mapped, under each approach. The pseudocode in Figure 6 describes how values are assigned to the a, h and ω variables that formally define each notional processor under either flat (lines 6–21) or semi-partitioned mapping (lines 23–54).

Under flat mapping, this derivation is conducted for each notional processor in turn, in order of increasing index (loop at line 6). Physical processors are also "filled up" in order of increasing index (with the variable $last\_h$ denoting the index of the "current" physical processor). Timeslot slices on each processor are allocated from left to right (along the time axis), in a repeating pattern. The variable $last\_\omega$ tracks what fraction of a timeslot (from left to right) is already filled up on the current physical processor; the next notional processor is mapped from there onwards. If it cannot be accommodated on the remaining capacity of the current physical processor (line 14), it uses as much as possible from it (line 15) and as much as needed from the next physical processor (line 15) and $last\_h$ is incremented (line 17).

Semi-partitioned mapping, in contrast, first maps notional processors 1 to m, each on a different physical processor (lines 23–30) using a single reserve occupying the *rightmost* part of the respective timeslot (lines 27–28). Remaining notional processors $m+1$ to m'' are then mapped (lines 33–54), in order of increasing index. In this second stage (as with flat mapping) physical processors are revisited (and their remaining capacity filled up) in order of increasing index, with the respective free timeslot slices being filled from left to right. The variable *acc* tracks the processor utilisation that has already been allocated to the current notional processor (over one or more physical processors). The variable *spent* tracks the utilisation of the current physical processor that has been assigned to (previous) notional processors indexed $m+1$ or higher (i.e. the left-hand side allocated portion of its timeslot, filled from left to right). The right-hand side allocated portion of its timeslot (allocated to a notional processor indexed 1 to m) is given by $1-\text{inflate}(U[last\_h])$. If the free timeslot slice in the middle (of length $(1-\text{inflate}(U[last\_h])-spent)\cdot S$) suffices for the yet unmapped portion of the (inflated) utilisation of the current notional processor, it is used from left to right as much as needed (lines 48–51); we can then proceed to map the next physical processor. Else, it is used up entirely (lines 41–43) and we move to the next physical processor (lines 44–46) and so on, until the current notional processor is fully accommodated.

```
1.  procedure map_notional_processors() is
2.    last_h:=1;
3.    last_a:=0;
4.    last_ω:=0;
5.    if (MAPPING_MODE=="FLAT") then
6.      for np:=1 to m" do
7.        ω[np]:=(last_ω+last_a) modulo 1;
8.        last_ω:=ω[np];
9.        a[np][0]:=0;
10.       h[np][0]:=last_h;
11.       if (ω[np]+inflate(U[np])<1) then
12.         a[np][1]:=inflate(U[np]);
13.         last_a:=a[np][1];
14.       else //spans two CPUs
15.         a[np][1]:=1-ω[np];
16.         h[np][1]:=last_h+1;
17.         last_h:=last_h+1;
18.         a[np][2]:=inflate(U[np]);
19.         last_a:=a[np][2];
20.       end if
21.     end for
22.   else //(MAPPING_MODE=="SEMI-PARTITIONED")
23.     for np:=1 to m do
24.       a[np][0]:=0;
25.       a[np][1]:=inflate(U[np]);
26.       h[np][0]:=np;
27.       tmp:=last_ω+1-inflate(U[np]);
28.       ω[np][0]:=fractional_part_of(tmp);
29.       last_ω:=ω[np][0];
30.     end for
31.     last_ω:=0; //rewind;
32.     spent:=0; //rewind;
33.     for np:=m+1 to m'' do
34.       a[np][0]:=0;
35.       h[np][0]:=last_h;
36.       z:=1;
37.       acc:=0;
38.       ω[np]:=last_ω; //initialise
39.       while (acc<U[np]) do
40.         if (acc+1-inflate(U[last_h])-spent<inflate(U[np]))then
41.           a[np][z]:=acc+(1-inflate(U[last_h]))-spent;
42.           h[np][z]:=last_h;
43.           acc:=a[np][z];
44.           z:=z+1;
45.           last_h:=last_h+1;
46.           spent:=0;
47.         else
48.           a[np][z]:=inflate(U[np]);
49.           h[np][z]=last_h;
50.           last_ω:=(last_ω+inflate(U[np])) modulo 1;
51.           spent:=a[np][z]-a[np][z-1];
52.         end if
53.       end while
54.     end for
55.   end if
56. end procedure
```

Fig. 6 Notional processor implementation.

Whether flat or semi-partitioned mapping is used, the sufficient and necessary condition for schedulability with NPS-F (i.e. mapping notional to physical processors) is

$$\sum_{p=1}^{m''} \text{inflate}(U_p) \leq m \tag{5}$$

3.1 Utilisation bound

Our proof for the utilisation bound of the algorithm (Theorem 3), relies on Theorem 1 and Lemma 1, which follow:

Theorem 1 *Assume First-Fit bin-packing with item sizes in the range (0, 1] over an infinite set of bins* $\{b_1,\ b_2,\ \ldots\}$. *Let* m'' *denote the index of the highest-indexed utilised bin, after bin-packing. Let* U_p *denote the utilised capacity of bin* b_p, $\forall p \in \{1, 2, \ldots\}$. *If* $m'' \geq 2$, *it holds that*

$$\frac{\sum_{p=1}^{m''} U_p}{m''} > \frac{1}{2} \tag{6}$$

Theorem 1 follows from (Garey and Johnson, 1979, p. 125).

Theorem 2 *For any set* $\mathbf{U} = \{U_1, U_2, \cdots, U_{m''}\}$ *of (not necessarily distinct) real numbers in the range (0,1],*

$$\sum_{p=1}^{m''} \text{inflate}(U_p) \leq m'' \cdot \text{inflate}\left(\frac{\sum_{p=1}^{m''} U_p}{m''}\right)$$

Proof If m''=1, the proof is trivial. If $m'' \geq 2$, let q, r be integers such that $U_q = \min_{p=1}^{m''}\{U_p\}$ and $U_r = \max_{p=1}^{m''}\{U_p\}$.

The function inflate is continuous and infinitely differentiable and $\frac{d}{dU}\text{inflate}(U) > 0$ and $\frac{d^2}{dU^2}\text{inflate}(U) < 0$, $\forall U \in [0, 1]$. Therefore,

$$2 \cdot \text{inflate}\Big(\frac{U_q + U_r}{2}\Big) \geq \text{inflate}(U_q) + \text{inflate}(U_r)$$

Let us obtain a modified set $\mathbf{U}^{(1)} = \{U_1^{(1)}, U_2^{(1)}, \cdots, U_{m''}^{(1)}\}$ by setting U_q and U_r equal to $\frac{U_q+U_r}{2}$. Then, $\sum_{p=1}^{m''} \text{inflate}(U_p^{(1)}) \geq \sum_{p=1}^{m''} \text{inflate}(U_p)$.

By repeating the procedure forever, each time with different q, r for the resulting modified set $\mathbf{U}^{(2)}$, $\mathbf{U}^{(3)}$,..., we converge to the set $\mathbf{U}^{(\infty)}$ with $U_1^{(\infty)} = U_2^{(\infty)} = \ldots = U_{m''}^{(\infty)} = \frac{1}{m''} \cdot \sum_{p=1}^{m''} U_p$. Then, since $\forall k$: $\sum_{p=1}^{m''} \text{inflate}(U_p^{(k+1)}) \geq \sum_{p=1}^{m''} \text{inflate}(U_p^{(k)})$, it holds that

$$\sum_{p=1}^{m''} \text{inflate}(U_p^{(\infty)}) \geq \sum_{p=1}^{m''} \text{inflate}(U_p) \Rightarrow$$

$$\sum_{p=1}^{m''} \text{inflate}(U_p) \leq m'' \cdot \text{inflate}\Big(\frac{1}{m''} \cdot \sum_{p=1}^{m''} U_p\Big) \tag{7}$$

□

Lemma 1 $\alpha(U) < \frac{1}{2\cdot\delta+1} \cdot U,\ \forall U \in (\frac{1}{2}, 1]$

Proof The function α is strictly decreasing and non-negative over $[\frac{1}{2}, 1]$. Thus, for any $U \in (\frac{1}{2}, 1]$, it holds that:

$$\frac{\alpha(U)}{U} < \frac{\alpha(\frac{1}{2})}{\frac{1}{2}} = \frac{1}{2\cdot\delta+1} \Rightarrow \alpha(U) < \frac{1}{2\cdot\delta+1} \cdot U \tag{8}$$

□

Theorem 3 *The utilisation bound of NPS-F is* $\frac{2\cdot\delta+1}{2\cdot\delta+2}$.

Proof An equivalent claim is that every task set with cumulative utilisation not above $\frac{2\cdot\delta+1}{2\cdot\delta+2} \cdot m$ is schedulable by NPS-F over m physical processors. This, we will prove.

Recall (Inequality 5) that the m'' notional processors can be mapped to m physical processors if and only if $\sum_{p=1}^{m''} \text{inflate}(U_p) \leq m$. Also, from Theorem 2:

$$\sum_{p=1}^{m''} \text{inflate}(U_p) \leq m'' \cdot \text{inflate}\left(\frac{\sum_{p=1}^{m''} U_p}{m''}\right) \tag{9}$$

Therefore, for schedulability, it suffices that

$$m'' \cdot \text{inflate}\left(\frac{\sum_{p=1}^{m''} U_p}{m''}\right) \leq m \tag{10}$$

If m''=1, the condition is trivially met. If m''≥2, let $\bar{U}$ denote $\frac{\sum_{p=1}^{m''} U_p}{m''}$. Then, Inequality 10 becomes:

$$m'' \cdot \text{inflate}\left(\bar{U}\right) \leq m \Leftrightarrow m'' \cdot (\bar{U} + \alpha(\bar{U})) \leq m \tag{11}$$

From Theorem 1: $\bar{U} > \frac{1}{2}$. Thus, by applying Lemma 1 to Inequality 11, it suffices for schedulability that

$$m'' \cdot (\bar{U} + \frac{1}{2\cdot\delta+1} \cdot \bar{U}) \leq m \Leftrightarrow m'' \cdot \bar{U} \cdot \frac{2\cdot\delta+2}{2\cdot\delta+1} \leq m \Leftrightarrow$$
$$m'' \cdot \bar{U} \leq \frac{2\cdot\delta+1}{2\cdot\delta+2} \cdot m \tag{12}$$

But $m'' \cdot \bar{U}$ equals the cumulative utilisation of the task set. Therefore, any task set of cumulative utilisation up to $\frac{2\cdot\delta+1}{2\cdot\delta+2} \cdot m$ is schedulable, which proves the theorem. □

3.2 Domination over partitioning with First-Fit bin-packing

We will show that NPS-F dominates partitioning with First-Fit bin-packing, in terms of the task sets that it can schedule.

Lemma 2 *Any task set schedulable using partitioning and First-Fit bin-packing is also schedulable by NPS-F (irrespective of the value for δ).*

Proof We will rely on the fact that the same First-Fit bin-packing used for partitioning, is also used within NPS-F, to populate the bins.

If a task set can be scheduled during First-Fit partitioning then, using NPS-F (and assuming the order of the tasks does not change), the same task-set, after the bin-packing step, will leave no more than m bins occupied. (In fact even the the task assignments will be the same, in both cases.) Then,

$$\sum_{i=1}^{m''} \text{inflate}(U_p) = \sum_{i=1}^{m} \text{inflate}(U_p) \leq \sum_{i=1}^{m} 1 = m \tag{13}$$

which means that the task set is schedulable by NPS-F. This holds for any value used for δ. □

In practice, with NPS-F, the designer can check whether $m'' \leq m$ after the bin-packing stage and, if so, has the option of using pure partitioning, instead of notional processors, to schedule the task set. This way, the additional preemptions associated with the implementation of the reserves are avoided.

3.3 Upper bound on preemptions

Definition 1 A task with outstanding computation at time t, is said to be preempted at time t if it executes on processor p just before t but not just after t.

By this definition, which we believe captures the notion of preemption used in the research community, a job that starts executing is not preempted, nor is one that finishes executing. Also, a job executing both just before and just after t but on different processors is, by the same definition, preempted at time t. Such a preemption is a *migration*.

With flat mapping, there exist two types of preemptions (other than due to task arrivals):

- type-α_1: Occurring when the reserve implementing a notional processor runs out (while a task is executing on it). At most one such preemption per notional processor per timeslot occurs. It is potentially a migration.
- type-β_1: Occurring when a notional processor (with a task currently executing on it) migrates to another physical processor. At most one such preemption per physical processor per timeslot occurs. For the preempted task, it is a

migration (unless the notional processor migration coincides with a context switch on the migrated notional processor and, additionally, the preempted task next resumes execution at a future instant when the notional processor has moved back to the original physical processor).

With semi-partitioned mapping, there likewise exist two types of preemptions (other than due to task arrivals):

- type-α_2: Occurring when the reserve implementing a notional processor runs out (while a task is executing on it). At most one such preemption per notional processor per timeslot occurs. For notional processors $\tilde{P}_1$ to $\tilde{P}_m$ it is never a migration (because they only utilise one physical processor each).
- type-β_2: Occurring whenever a notional processor (with a task currently executing on it) migrates to another physical processor. At most one such preemption per physical processor per timeslot occurs. It is a migration (except under the conditions earlier described in the context of type-β_1 preemptions.

Thus, during an interval of length Δt, overall preemptions (including migrations) cannot exceed

$$N_{pr}(\Delta t) = N_{arr}(\Delta t) + \underbrace{\left\lceil \frac{\Delta t}{S} \right\rceil \cdot m''}_{\text{type-}\alpha} + \underbrace{\left\lceil \frac{\Delta t}{S} \right\rceil \cdot m}_{\text{type-}\beta}$$
$$= N_{arr}(\Delta t) + \left\lceil \frac{\Delta t}{S} \right\rceil \cdot (m + m'') \quad (14)$$

where $N_{arr}(\Delta t)$ is an upper bound on task arrivals within the interval. We next eliminate m'' from Equation 14:

Corollary 1 *For any task set τ with $U_\tau \leq 1$: $m'' < 2 \cdot m$*

Proof From Theorem 1 follows that $\sum_{p=1}^{m''} U_p > \frac{m''}{2}$. But $\sum_{p=1}^{m''} U_p = \sum_{i=1}^{n} \frac{C_i}{T_i} = m \cdot U_\tau \leq m$. Therefore, $m > \frac{m''}{2} \Rightarrow m'' < 2 \cdot m$ □

Thus from Equation 14 and Corollary 1, we obtain:

$$N_{pr}(\Delta t) < N_{arr}(\Delta t) + \left\lceil \frac{\Delta t}{S} \right\rceil \cdot 3 \cdot m \Rightarrow$$
$$N_{pr}(\Delta t) < N_{arr}(\Delta t) + \left\lceil \frac{\Delta t}{\text{TMIN}} \right\rceil \cdot 3 \cdot m \cdot \delta \quad (15)$$

This bound for preemptions is the same as for the variant of EKG for sporadic tasks (Andersson and Bletsas, 2008), which also uses a parameter δ, with the same semantics as in NPS-F (i.e. trading off schedulable utilisation vs preemptions). However, for the same δ, NPS-F always has higher utilisation bound (see Table 1 and, for proof, Theorem 6 in the Appendix). Moreover, the algorithm in (Andersson and Bletsas, 2008) caps processor utilisation at its utilisation bound (except when assigning tasks with very large utilisations). NPS-F thus dominates that scheme without causing more preemptions.

UB	$\delta = 1$	$\delta = 2$	$\delta = 3$	$\delta = 4$	$\delta \rightarrow \infty$
NPS-F	75%	83.3%	87.5%	90%	$\rightarrow$100%
EKG-spor. 2008	65.7%	79.8%	85.6%	88.9%	$\rightarrow$100%

Table 1 Comparison of utilisation bounds

Yet a comparison, in terms of preemptions, of NPS-F with the original NPS (Bletsas and Andersson, 2009a) is more complicated. NPS has a utilisation bound of $66.\bar{6}\%$, lower than that of even NPS-F with δ=1. Yet its preemption bound is seemingly lower than that of NPS-F. For a given task set schedulable by both algorithms, are preemptions under NPS fewer than under NPS-F with $\delta = 1$?

The answer is that it depends on the task set. Indeed, NPS can, retroactively, be described, as NPS-F with semi-partitioned mapping and $\delta = 1$ with two differences: (D1) different bin-packing and (D2) sub-optimal sizing of notional processors indexed $m + 1$ or higher. D1 alone renders comparisons dependent on the actual task set. However, D2 affects *typical* behavior, as we will explain:

For NPS, the upper bound on preemptions is

$$N_{pr}(\Delta t) = N_{arr}(\Delta t) + \left\lceil \frac{\Delta t}{\text{TMIN}} \right\rceil \cdot \left(2 \cdot m + \frac{m}{3}\right)$$

which appears lower than the respective upper bound for NPS-F (see Inequality 15). However, it also holds that, for both NPS and NPS-F($\delta = 1$),

$$N_{pr}(\Delta t) = N_{arr}(\Delta t) + \left\lceil \frac{\Delta t}{\text{TMIN}} \right\rceil \cdot (m + m'')$$

where m'' is the number of notional processors[1].

It is because of the fact that m'' could not exceed $m + \lceil \frac{m}{3} \rceil$ under NPS, due to its relative inefficiency (D2) that NPS appears more preemption-light than NPS-F(δ=1). It is this same inefficiency that limits the utilisation bound of NPS to 66.6% (vs 75% for NPS-F(δ=1)). Inversely reasoning, due to NPS-F being more efficient in utilising processing capacity, it is likelier that fewer bins are needed to accommodate the same task set under NPS-F(δ=1) compared to NPS, than vice versa. In turn, this would mean fewer preemptions. In this light, it is not correct to say that NPS-F(δ=1) is more preemption-intensive than NPS.

Regarding comparisons with EDF-WM, Kato et al (2009), in their experiments with EDF-WM and EDF-SS established that even EDF-SS($\delta = 1$) typically involves many more context switches (and, obviously preemptions) than EDF-WM – up to an order of magnitude. Since EDF-SS (in the context

[1] Note that the terminology differs slightly in (Bletsas and Andersson, 2009a). In (Bletsas and Andersson, 2009a), where semi-partitioned mapping is used, notional processors indexed 1 to m are conflated with the physical processors to which they are mapped and are not explicitly termed notional.

of implicit-deadline tasks) and NPS-F have essentially the same upper bound on preemptions due to the implementation of the reserves (which constitute the bulk of preemptions), as a function of δ and the time interval length, the unfavorable comparison would also extend to NPS-F.

Therefore NPS-F is not the most preemption-light multiprocessor scheduling scheme, among all those that typically perform well – but only among those with UB$>$ 50%.

4 Clustered variant of NPS-F

We now introduce a derivative of NPS-F, motivated by desire to (i) adapt scheduling approaches to the state of the art in chip design, i.e. multicores; (ii) alleviate a weakness, i.e. that migrations, though limited, may be costly.

4.1 On multicores and processor clusters

Multicore chips are by now mainstream and common – even in embedded systems (ARM Ltd, 2008). The major manufacturers already offer affordable dual-,triple- (AMD Inc., 2008a), quad-, six- (Intel Corporation, 2008c) and eight-cores. Common to most latest designs is a top-level cache shared by all cores (Intel Corporation, 2008a,c,b, 2009; AMD Inc., 2008c,b,a).

In such architectures, reads and writes on the the same data by different cores of the same chip, necessitate less main memory I/O, compared to cores from different chips. In particular, when a task migrates from core P_1 to core P_2 on the same chip, the task state that P_2 needs is already on-chip. This is important for performance because top-level cache misses, by causing accesses to main memory, impact performance far more than do lower-level cache misses (Fedorova et al, 2005; Anderson et al, 2006). Indeed, were P_1 and P_2 on different chips, top-level cache misses would be likely.

We leverage this by dividing processors into (non-overlapping) clusters, independently scheduled under NPS-F. Each cluster is formed by the cores of a given respective chip. This ensures that off-chip migrations never occur. Therefore, our approach does not cause additional top-level misses (and associated main memory I/O). Performance losses due to task migration are thus kept in check. In the general case, eliminating off-chip migrations also helps handle other issues: cache coherency, locking, per-processor OS data structures.

The concept of processor clusters is well-known in the the literature, including recent work (Brandenburg et al, 2008; Shin et al, 2007; Calandrino et al, 2007). Calandrino et al (2007) share motivation with us (experimenting with EDF-scheduled clusters over multicores, so as to reduce migration and preemption costs relative to global EDF). Yet, the utilisation bound in (Calandrino et al, 2007) is just 50%. Shin et al (2007) aim to improve schedulability but allow for overlapping clusters (which runs counter to our motivation); also, no utilisation bounds are proven.

4.2 The modified algorithm

Let μ (a divisor of m) denote the cluster size. We thus have $\frac{m}{\mu}$ clusters, Q_1 to $Q_{\frac{m}{\mu}}$, of μ processors each. Cluster Q_q is formed by processors $P_{(q-1)\cdot\mu+1}$ to $P_{(q-1)\cdot\mu+\mu}$.

All that our clustered algorithm does is partition the task set into subsets, each of which can provably be scheduled over one cluster using NPS-F. To describe it, it thus suffices to explain how this partitioning is performed – follow the pseudocode of Figure 7 in parallel with our comments.

To each cluster Q_q corresponds a different set of bins ($b_1^{(q)}$, $b_2^{(q)}$, $b_3^{(q)}$, ...). Tasks are assigned one by one, with clusters tested for assignment in order of index. Inequality 5 (sufficient and necessary condition for schedulability under NPS-F) becomes in the context of a cluster:

$$\sum_{p=1}^{m''^{(q)}} \text{inflate}(U_p^{(q)}) \leq \mu \tag{16}$$

If a task can be assigned First-Fit to (one of) the bins of the cluster in consideration while satisfying Inequality 16, it is assigned there; else, the next cluster is considered. Therefore, post-assignment, all clusters are schedulable. Note, however, that in a generalisation of previous semantics, during bin-packing, $m''^{(q)}$ refers to the number of utilised bins pertaining to cluster Q_q *at the given point* in the execution of the algorithm and, after every assignment, its value is updated accordingly.

Note also that, although infinite bins correspond to each cluster, only a finite number thereof are potential assignment targets during bin-packing. If, prior to attempting to assign some task τ_i within some cluster Q_k, only bins $b_1^{(q)}$ to $b_k^{(q)}$ of Q_q have tasks already assigned to them, we then need only test bins $b_1^{(q)}$ to (at most) $b_{k+1}^{(q)}$ as assignment targets for τ_i. If it cannot be assigned to $b_{k+1}^{(q)}$ (as the only task there) *while also satisfying Inequality 16*, this would also hold for $b_{k+2}^{(q)}$ onwards. We can thus move on to Q_{q+1}.

Our description of (non-clustered) NPS-F, did not rely on any particular ordering of tasks during bin-packing. However, for the clustered variant, we assume that tasks with utilisation $\frac{2\cdot\delta+1}{2\cdot\delta+2} \cdot \frac{\mu}{\mu+1}$ or higher (i) are indexed in order of decreasing utilisation and (ii) precede all tasks with utilisation below $\frac{2\cdot\delta+1}{2\cdot\delta+2} \cdot \frac{\mu}{\mu+1}$. This allows for a higher utilisation bound than would be possible if task ordering were arbitrary (as will be seen in the proof of Theorem 4).

We use the notation NPS-F$_{m:\mu}$ for the clustered algorithm, meaning that we have $\frac{m}{\mu}$ clusters of μ processors each, with each cluster scheduled by NPS-F. Non-clustered NPS-F can thus be described as NPS-F$_{m:m}$. By UB$_{m:\mu}$, we denote the utilisation bound of NPS-F$_{m:\mu}$. In notation, we can also use fixed values in place of m and μ.

```
1.  for i:=1 to n do //tasks already ordered
2.    unassigned:=TRUE;
3.    q:=1;
4.    while (unassigned==TRUE) do
5.      assign τ_i First-Fit over b_1^(q), b_2^(q), b_3^(q), ...
        so that Ineq. 16 is satisfied.
6.      if (task was assigned in line 5) then
7.        unassigned:=FALSE;
9.      else
10.       if (q==m/μ) then //last cluster
11.         exit(FAILURE);
12.       else
13.         q:=q+1; //next cluster
14.       end if
15.     end if
16.   end while
17. end for
```

Fig. 7 Bin-packing over clusters

4.3 Utilisation bound

Theorem 4 $\mathrm{UB}_{m:\mu} \not\leq \frac{2\cdot\delta+1}{2\cdot\delta+2} \cdot \frac{\mu}{\mu+1}\ \forall \mu \geq 2$

Proof We will prove the claim by showing that every task set with $U_\tau \leq \frac{2\cdot\delta+1}{2\cdot\delta+2} \cdot \frac{\mu}{\mu+1}$ is schedulable.

Assume that an unschedulable task set with $U_\tau \leq \frac{2\cdot\delta+1}{2\cdot\delta+2} \cdot \frac{\mu}{\mu+1}$ exists. Then, some task τ_f with utilisation u_f will fail to be assigned to some cluster, subject to existing assignments. We explore two mutually exclusive cases:

- $u_f > \frac{2\cdot\delta+1}{2\cdot\delta+2} \cdot \frac{\mu}{\mu+1}$:
  Then, due to the task ordering, all tasks previously assigned had utilisations no less than $u_f > \frac{2\cdot\delta+1}{2\cdot\delta+2} \cdot \frac{\mu}{\mu+1} \geq \frac{1}{2}$ – and each is the only task in its bin. Moreover, on every cluster at least μ such tasks are already assigned (or, from Inequality 5 the assignment of τ_f would not have failed). Thus, on every cluster Q_q the cumulative utilisation of tasks already assigned exceeds $\mu \cdot \frac{2\cdot\delta+1}{2\cdot\delta+2} \cdot \frac{\mu}{\mu+1}$ before attempting to assign τ_f.
- $u_f \leq \frac{2\cdot\delta+1}{2\cdot\delta+2} \cdot \frac{\mu}{\mu+1}$:
  Since τ_f could not be assigned, we deduce that, on every cluster Q_q, the cumulative utilisation of tasks already assigned exceeds $\mu \cdot UB_{\mu:\mu}$, if incremented by u_f (or τ_f would have been assigned). Therefore (from Theorem 3) the cumulative utilisation of tasks already assigned before the attempt to assign τ_f exceeds

$$\mu \cdot \frac{2\cdot\delta+1}{2\cdot\delta+2} - u_f \geq \mu \cdot \frac{2\cdot\delta+1}{2\cdot\delta+2} - \frac{2\cdot\delta+1}{2\cdot\delta+2} \cdot \frac{\mu}{\mu+1} = \mu \cdot \frac{2\cdot\delta+1}{2\cdot\delta+2} \cdot \frac{\mu}{\mu+1}$$

In any case, for every cluster Q_q, the cumulative utilisation of tasks already assigned to Q_q exceeds $\mu \cdot \frac{2\cdot\delta+1}{2\cdot\delta+2} \cdot \frac{\mu}{\mu+1}$. Therefore, the entire system is utilised by more than $\frac{2\cdot\delta+1}{2\cdot\delta+2} \cdot \frac{\mu}{\mu+1}$ even before attempting to assign τ_f. This contradicts the initial assumption that $U_\tau \leq \frac{2\cdot\delta+1}{2\cdot\delta+2} \cdot \frac{\mu}{\mu+1}$. □

UB	$\delta = 1$	$\delta = 2$	$\delta = 3$	$\delta = 4$	$\delta \to \infty$
$UB_{m:2}$	50%	55.5%	58.3%	60%	→66.6%
$UB_{m:3}$	56.25%	62.5%	65.625%	67.5%	→75%
$UB_{m:4}$	60%	66.6%	70%	72%	→80%
$UB_{m:6}$	64.2%	71.4%	75%	77.1%	→85.7%
$UB_{m:8}$	66.6%	74.0%	77.7%	80%	→88.8%
$UB_{m:16}$	70.5%	78.4%	82.3%	84.7%	→94.1̄%
$UB_{m:m}$	75%	83.3%	87.5%	90%	→100%
UB_{2008}	65.7%	79.8%	85.6%	88.9%	→100%

Table 2 Utilisation bounds of $NPS_{m:\mu}$ (for various μ) vs non-clustered NPS-F and (Andersson and Bletsas, 2008)

By inspection, the utilisation bounds of NPS-$F_{m:\mu}$ and non-clustered NPS-F converge, as μ increases. Therefore, as cores per chip increase in the near future, NPS-$F_{m:\mu}$ becomes increasingly practical and attractive. Table 2 compares the utilisation bound of NPS-$F_{m:\mu}$ for different μ and δ with that of non-clustered NPS-F and the algorithm by Andersson and Bletsas (2008) (also suffering from off-chip task migrations).

At present, we view $\mu = 4$ as the most relevant cluster size (given current chip offerings). This motivates one optimisation to the algorithm, which raises its utilisation bound for $\delta = 1$ (the most interesting setting, in our view, by virtue of being the most preemption-light) to $\frac{5}{8} = 62.5\%$ (up from 60%). This is achieved merely by a more restrictive task ordering, wherein tasks with utilisation $\frac{1}{2}$ or higher (i) are indexed in order of decreasing utilisation and (ii) precede all other tasks. For proof, see the Appendix (Theorem 7).

4.4 Upper bound on preemptions

Each of the $\frac{m}{\mu}$ clusters is independently scheduled under (non-clustered) NPS-F. Therefore, preemptions on cluster Q_q within an interval of Δt time units are bounded by

$$N_{pr}^{q}(\Delta t) = N_{arr}^{q} + \left\lceil \frac{\Delta t}{\text{TMIN}} \right\rceil \cdot 3 \cdot \mu \cdot \delta \tag{17}$$

from Equation 15 (via substitution of m by μ). Over the entire system, preemptions are thus bounded by

$$N_{pr}^{m:\mu} = \sum_{q=1}^{\frac{m}{\mu}} N_{pr}^{q}(\Delta t) = \sum_{q=1}^{\frac{m}{\mu}} N_{arr}^{q}(\Delta t) + \sum_{q=1}^{\frac{m}{\mu}} \left(\left\lceil \frac{\Delta t}{\text{TMIN}} \right\rceil \cdot 3 \cdot \mu \cdot \delta \right)$$

$$= N_{arr}(\Delta t) + \left\lceil \frac{\Delta t}{\text{TMIN}} \right\rceil \cdot 3 \cdot \mu \cdot \delta \cdot \frac{m}{\mu} = N_{arr}(\Delta t) + \left\lceil \frac{\Delta t}{\text{TMIN}} \right\rceil 3 \cdot m \cdot \delta \tag{18}$$

This bound is the same as that for non-clustered NPS-F (Equation 15) and is not dependent on μ. In reality though, we need not use the same timeslot

length on all clusters. Each cluster Q_q could use a (hopefully longer) timeslot

$$S_q = \frac{1}{\delta} \cdot \min_{\substack{\tau_i \text{ assign-} \\ \text{ed to } Q_q}} T_i$$

instead of $\frac{1}{\delta} \cdot \text{TMIN}$. This optimisation has no downside and will reduce preemptions in most cases. In fact, it cures another weakness of NPS-F, also present in (Andersson and Bletsas, 2008): that, a single task with too short an interarrival time would force an accordingly short timeslot (leading to numerous preemptions). With per-cluster timeslot selection, the effect is localised to one cluster.

5 Omega Optimisation

According to our definitions so far, if a notional processor uses multiple physical processors, then it employs temporally adjacent reserves on these processors. It is possible however, to relax this definition so that a notional processor can be mapped to temporally non-adjacent reserves (which, however, still have to be temporally non-overlapping). The rationale for such an implementation is to allow more efficient use of processing capacity in the average case (without compromising the utilisation bound we proved earlier). We formulate this optimisation in the context of non-clustered NPS-F; its propagation to clustered NPS-F is straightforward.

5.1 Motivation

Suboptimal use of processing capacity may result from requiring that the reserves of a notional processor always be temporally adjacent. This can be demonstrated via the following example:

Example 1 A set τ of three tasks, with utilisations of $\frac{5}{9}$, $\frac{8}{17}$ and $\frac{5}{9}$, is to be scheduled over two processors, P_1 and P_2, using NPS-F with $\delta = 1$.

Using NPS-F with $\delta = 1$, each task is assigned (First-Fit) to a different notional processor. The processing capacity required for implementing notional processors $\tilde{P}_1$ and $\tilde{P}_3$ is inflate($\frac{5}{9}$) = $\frac{5}{7}$, whereas for $\tilde{P}_2$ it is inflate($\frac{8}{17}$) = $\frac{16}{25}$. If summed up, these requirements exceed the processing capacity of the two physical processors – a hypothetical third processor P_3 would be required for schedulability (see Figure 8(a)).

Yet, if the reserve for $\tilde{P}_2$ on P_3 is shifted to the right, along the time axis, it can then be shortened so that both (i) the workload of $\tilde{P}_2$ is schedulable (despite utilising less processing capacity) and (ii) $\tilde{P}_2$ can be implemented entirely on processor $\tilde{P}_3$. Figure 8(b) depicts the values for this offset and for the sizes of the reserves. (That $\tilde{P}_2$ is schedulable can be seen by the fact that over any time window of length S or more, $\tilde{P}_2$ receives no fewer units

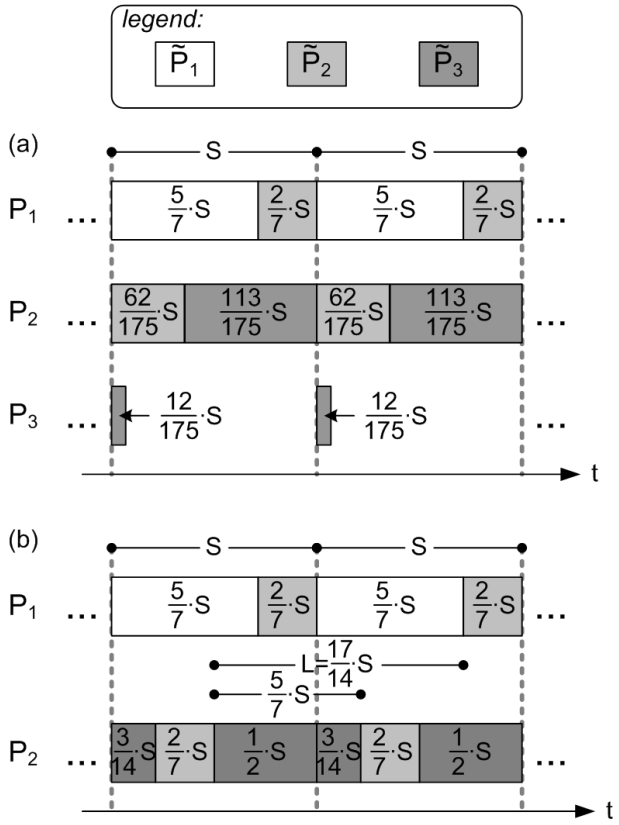

Fig. 8 The Ω-optimisation can result in more efficient usage of processor capacity.

of processor time than it does during the same interval in the previous case (Figure 8(b)).

Intuitively, by breaking up the back-to-back execution of the two reserves for $\tilde{P}_2$, we made the supply of processor time more fine-grained, i.e. better approximating a linear function of time. In turn, this reduced the need for inflation. Thus we were able to shorten the reserve of $\tilde{P}_2$ on P_2, using up less processor capacity.

Our optimisation lies in leveraging this effect.

5.2 Problem formulation

Consider a task set $\tau \stackrel{\text{def}}{=} \{\tau_1, \tau_2, \ldots \tau_n\}$ to be scheduled over physical processors P_1 to P_m using NPS-F. We assume flat mapping of notional to physical processors. However, in a deviation from the original NPS-F algorithm, for any notional processor that utilises multiple reserves (each on a different physical processor) we will permit those reserves to not necessarily run back-to-back.

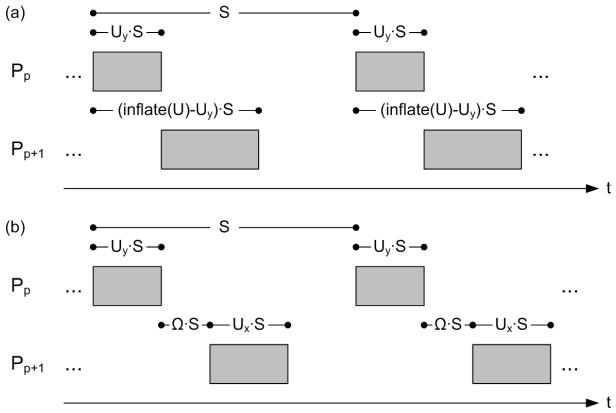

Fig. 9 Formulation of Ω-optimisation problem.

Under flat mapping, each notional processor is mapped to at most 2 physical processors. Assume that a notional processor serving tasks of cumulative utilisation U is split between two processors, P_p and P_{p+1}. Under the original algorithm the reserve on P_{p+1} starts as soon as the reserve on P_p ends. If the reserve on P_p is $U_y \cdot S$ time units long, then the reserve on P_{p+1} is dimensioned to (inflate$(U) - U_y) \cdot S$ time units (see Figure 9(a)), such that the joint duration of the two adjacent reserves is inflate$(U) \cdot S$.

However, if we break up the back-to-back execution of the reserves by introducing an offset of $\Omega \cdot S$ time units for the start of the reserve on P_{p+1}, relative, to the end of the reserve on P_{p+1}, the optimal size $(U_x \cdot S)$ of the reserve on P_{p+1} may differ. For the offset, it has to hold that

$$0 \leq \Omega \leq 1 - (U_y + U_x) \tag{19}$$

so as to avoid any temporal overlap of the reserve on P_{p+1} with the next reserve on P_p. The value for U_x has to be adequate for ensuring schedulability.

We will proceed with analysis on how to optimally derive U_x for any notional processor split between two physical processors, serving tasks of cumulative utilisation U, as a function of U and U_y (with $U_y \cdot S$ being the length of the reserve on the first processor of the two) and a given value for Ω.

Subsequently, we will use this analysis so as to derive the optimal value for scheduling parameter Ω. Ultimately, this allows us to derive the combination of Ω, U_x that minimises U_x while still ensuring schedulability.

5.3 Optimal derivation of U_x as a function of U, U_y, Ω

We first present a statement (generalisation of Theorem 5 earlier used and found in the Appendix) will prove handy later on:

Theorem: *It suffices, for a set τ' of implicit-deadline tasks with cumulative utilisation U to be schedulable under EDF, if, within any possible time*

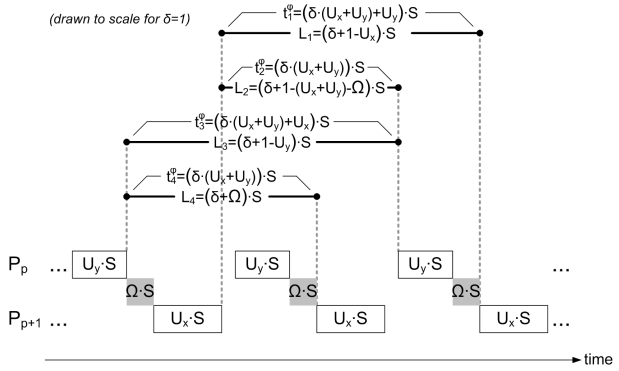

Fig. 10 Candidate time windows (among those no shorter than S) for minimal τ^{φ} as a fraction of their length.

interval of length $L \geq \min_{\tau_i \in \tau'} T_i$, the time t^{φ} available for the execution of τ' (possibly in disjoint time intervals) is $U \cdot L$ or greater.

The above claim is proven as Theorem 8 in the Appendix.

Now consider a notional processor serving tasks of cumulative utilisation U and split between physical processors P_p and P_{p+1}, via reserves of length $U_y \cdot S$ and $U_x \cdot S$ respectively, which are temporally separated by an interval of length $\Omega \cdot S$ (see Figure 9).

We wish to determine the lowest value for U_x for which the schedulability of the tasks assigned to the notional processor in consideration is ensured. For a time window of length L, let τ^{φ} denote the aggregate length of all subintervals overlapping with (one of) the two reserves. From Theorem 8 (and using $S = \frac{1}{\delta} \cdot \min T_i$), a sufficient condition for schedulability is:

$$\text{For every time window of length } L \geq \delta \cdot S\text{:} \quad \frac{\tau^{\varphi}}{L} \geq U \tag{20}$$

Equivalently, it suffices that $\frac{\tau^{\varphi}}{L} \geq U$ for the time window of length $L \geq \delta \cdot S$ for which the fraction $\frac{\tau^{\varphi}}{L}$ is minimised. By observation, such a time window would start at the end of some reserve and end at the start of some reserve for the notional processor in consideration. There are four combinations (depending on which of the two reserves it is, in either case), therefore there are four candidates for the time window with minimal $\frac{\tau^{\varphi}}{L}$. These are shown in Figure 10, annotated by their lengths (L_1 to L_4) and respective aggregate overlaps with reserves (τ_1^{φ} to τ_4^{φ}). Therefore, in any case, it suffices for schedulability, that the following Inequalities all hold:

$$U \leq \frac{\delta \cdot (U_x + U_y) + U_y}{\delta + 1 - U_x} \tag{21}$$

$$U \leq \frac{\delta \cdot (U_x + U_y)}{\delta + 1 - (U_x + U_y) - \Omega} \tag{22}$$

$$U \leq \frac{\delta(U_x + U_y) + U_x}{\delta + 1 - U_y} \tag{23}$$

$$U \leq \frac{\delta \cdot (U_x + U_y)}{\delta + \Omega} \tag{24}$$

Inequalities 21 to 24 can be respectively rewritten as:

$$U_x \geq \frac{(\delta + 1) \cdot (U - U_y)}{\delta + U} \tag{25}$$

$$U_x \geq \frac{U \cdot (\delta + 1 - \Omega)}{\delta + U} - U_y \tag{26}$$

$$U_x \geq U - \frac{(U + \delta) \cdot U_y}{\delta + 1} \tag{27}$$

$$U_x \geq \frac{U \cdot (\delta + \Omega)}{\delta} - U_y \tag{28}$$

Let F_1 to F_4 denote the right-hand sides of Inequalities 25 to 28 (in that order). Of these, only F_2 and F_4 are functions of Ω – strictly increasing/strictly decreasing, respectively. Therefore $\max(F_2, F_2)$ is minimal when

$$F_2 = F_4 \Leftrightarrow \frac{\delta + 1 - \Omega}{\delta + U} = \frac{\delta + \Omega}{\delta} \Leftrightarrow \Omega = \frac{\delta \cdot (1 - U)}{2 \cdot \delta + U} \tag{29}$$

Substituting this optimal Ω and rewriting yields:

$$U_x \geq U - U_y + (1 - U) \cdot \frac{U - U_y}{\delta + U} \tag{30}$$

$$U_x \geq U - U_y + (1 - U) \cdot \frac{U}{2 \cdot \delta + U} \tag{31}$$

$$U_x \geq U - U_y + (1 - U) \cdot \frac{U_y}{\delta + 1} \tag{32}$$

The set of constraints expressed by Inequalities 30 to 32 can be condensed into Equation 33, giving the optimal U_x as a function of U_y, U (to be used in combination with the optimal Ω earlier derived as a function of U_y, U).

$$U_x = U - U_y + (1 - U) \cdot \max\left(\frac{U - U_y}{\delta + U}, \frac{U}{2 \cdot \delta + U}, \frac{U_y}{\delta + 1}\right) \tag{33}$$

```
1.  procedure map_notional_processors() is
2.    last_h:=1;
3.    last_β:=0;
4.    last_ω:=0;
5.    //Ω-optimisation assumes flat mapping
6.    for np:=1 to m'' do
7.      ω[np]:=(last_ω+last_β) modulo 1;
8.      last_ω:=ω[np];
9.      a[np][0]:=0;
10.     h[np][0]:=last_h;
11.     if (ω[np]+inflate(U[np])<1) then
12.       β[np][0]:=inflate(U[np]);
13.       last_β:=β[np][1];
14.     else //spans two CPUs
15.       β[np][1]:=1-ω[np];
16.       h[np][1]:=last_h+1;
17.       last_h:=last_h+1;
18.       Ω:=right-hand side of Eq. 29
19.       Ux:=right-hand side of Eq. 33
20.       a[np][1]:=β[np][0]+Ω;
21.       β[np][1]:=a[np][1]+Ux;
22.       last_β:=β[np][1];
23.     end if
24.   end for
25. end procedure
```

Fig. 11 Ω-optimised notional processor implementation.

5.4 Implementing the optimisation

Having solved the problem of the Ω-optimisation, we next describe how it is implemented.

A notional processor formed by not necessarily temporally adjacent (but non-overlapping) reserves can be formally described by the (more general) notation

$$\big((a_0, a_1, ..., a_{z-1}), (\beta_0, \beta_1, ..., \beta_{z-1}), (h_0, h_1, ..., h_{z-1}), \omega\big)$$

with

- $0 = a_0 < a_1 < \ldots < a_{z-1} \leq 1$,
- $0 < \beta_0 < \beta_1 < \ldots < \beta_{z-1} \leq 1$,
- $a_u < \beta_u \leq a_{u+1},\ \forall u \in \{0, 1, \ldots, z-1\}$
- $h_u \in \{1, 2, \ldots, m\},\ \forall u \in \{\{0, 1, \ldots, z-1\}$
- $0 \leq \omega < 1$.

The semantics are that on time instant t, if $\exists u \in \{0, 1, \ldots, z-1\}$ such that $(t - \omega \cdot S)$ modulo $S \in [a_u, \beta_u)$, then the notional processor in consideration is mapped to processor P_{h_u}. (Note that for notional processors formed by temporally adjacent reserves, $\beta_u = a_{u+1}$.)

The pseudocode for the mapping of notional processors, updated with the Ω-optimisation, is presented in Figure 11.

5.5 Comparison with NPS-F

It can be easily shown that the Ω-optimised NPS-F (denoted NPS-FΩ) dominates unoptimised NPS-F (for the same δ). Indeed, the original NPS-F differs from NPS-FΩ in that it always (suboptimally) uses Ω=0 whenever splitting a notional processor between physical processors. Let usage($\tilde{P}_p$) denote the processor capacity used for implementing notional processor $\tilde{P}_p$ under NPS-FΩ. Then usage($\tilde{P}_p$) equals U_y+U_x for the respective notional processor, if mapped to two physical processors or inflate(U_p), if it uses just one physical processor.

From our analysis, inflate(U_p) $\leq$ usage($\tilde{P}_p$). Thus:

$$\tau \text{ schedulable by NPS-F} \Rightarrow \sum_{p=1}^{m''} \text{inflate}(U_p) \leq m \Rightarrow$$

$$\sum_{p=1}^{m''} \text{usage}(\tilde{P}_p) \leq m \Rightarrow \tau \text{ schedulable by NPS-F}\Omega \tag{34}$$

Therefore, the utilisation bound of NPS-FΩ is no less than that of NPS-F.

Regarding preemptions, NPS-FΩ may accommodate on m processors more notional processors than NPS-F (as in the example of Figure 8). Yet, Corollary 1 still holds for it. Hence the upper bound on preemptions expressed by Inequality 15 also holds for NPS-FΩ.

6 Experimental evaluation

Earlier, we characterised the performance of NPS-F via utilisation bounds. This metric accurately reflects performance guarantees *in the worst case*. We shall also evaluate (the variants of) NPS-F in terms of the success rate in scheduling task sets of a given utilisation (i.e. above the utilisation bound). Therefore, we experimentally evaluate (i) the effect of the various configurations on the performance of NPS-F (i.e. Ω-optimisation, clustering, task set ordering) and (ii) the performance of NPS-F against other multiprocessor scheduling algorithms for implicit-deadline tasks with utilisation bound at least 50% (including EDF-WM (Kato et al, 2009), which was not yet been published when the original NPS-F was conceived).

For the experiments we used the task set generator by Ted Baker[2]. This generator generates task sets for up to $m \leq 8$ processors (we chose $m = 2$, $m = 4$ and $m = 8$) with task set utilisations generates as pseudo-random variables conforming to different probability distributions:

Bimodal distribution: With a probability of 1/3, the task utilisation is chosen as a random variable uniformly distributed in the range [0.5, 1]. With a (complementary) probability of 2/3, it is chosen as a random variable uniformly distributed in the range [0, 0.05].

[2] We are grateful to Prof. Baker for granting us use of this tool.

Exponential distribution: The task utilisation is an exponentially distributed random variable with mean 0.5.

Uniform distribution: The task utilisation is a random variable uniformly distributed in the range [0,1].

Each task set is put into one of 100 "buckets", according to its quantised utilisation (e.g. all task sets with $U_\tau \in [0.77, 0.78)$ are put in the same bucket). For each bucket, we plot the fraction of task sets schedulable by each algorithm, according to the respective offline feasibility test[3].

The other algorithms in the performance comparison are: partitioned EDF, Ehd2-SIP, DM-PM, EDF-WM and the sporadic variant of EKG. The suffix /DU denotes task ordering by decreasing utilisation (which usually improves performance). The suffix /opt denotes the optimised partial task ordering for NPS-F$_{m:4}$ described earlier (Theorem 7). Otherwise, the algorithms use the tasks in whichever order they are to be found (unless they explicitly enforce some other default ordering).

The results appear as graphs, plotting the fraction of schedulable task sets for each "bucket" of quantised task set utilisation. For the utilisation ranges plotted, each bucket contains over 17000 task sets. To relieve visual congestion, we have separate graphs comparing NPS-F variants with each other and separate graphs comparing the best-performing configurations with the other algorithms.

6.1 Experiments for 2 processors

Figure 12 compares different variants/configurations of non-clustered NPS-F (clustering is not possible on 2 processors). For all three distributions, the Ω-optimisation offers no noticeable improvement, over 2 processors. Conversely, task ordering by decreasing utilisation, does. The effect of parameter δ is also explored (for the best-performing configuration, NPS-FΩ/DU), for small values of δ (which are the most interesting, due to the lower preemption counts).

Figure 13 shows that, over 2 processors, NPS-F$\Omega(\delta = 1)$ barely performs any better than partitioning. EDF-WM/DU clearly outperforms other algorithms, even though it does not have a proven utilisation bound above 50%. It also involves fewer preemptions than NPS-F$\Omega(\delta = 1)$.

6.2 Experiments for 4 processors

Figures 14 and 15 present the respective experimental results for 4 processors.

[3] We wish to thank Dr Shinpei Kato, for providing us, upon request, with code implementing the schedulability tests for his algorithms, DM-PM and EDF-WM.

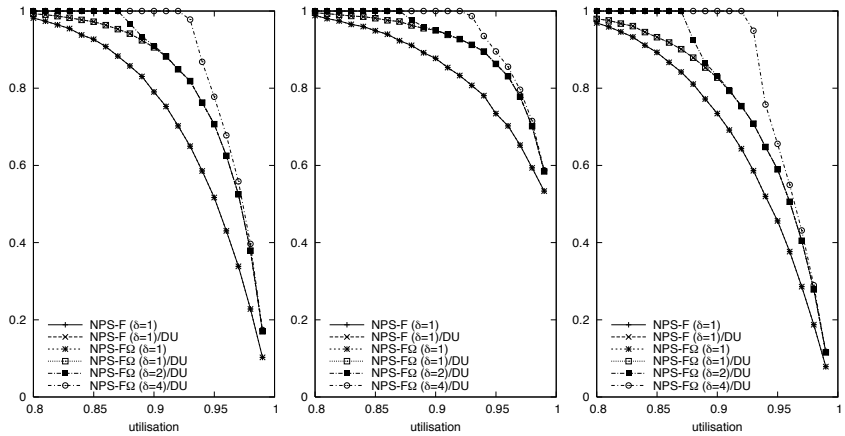

Fig. 12 Fraction of schedulable task sets of a given utilisation range, over 2 processors. Task utilisations conform to the bimodal/exponential/uniform distribution (left to right).

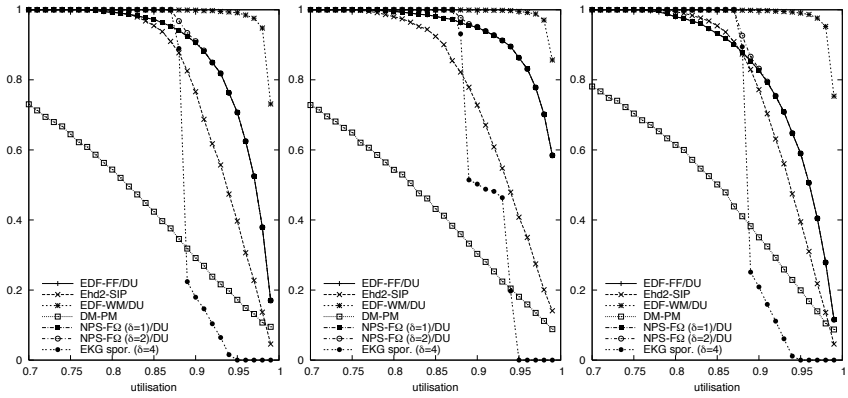

Fig. 13 Fraction of schedulable task sets of a given utilisation range, over 2 processors. Task utilisations conform to the bimodal/exponential/uniform distribution (left to right).

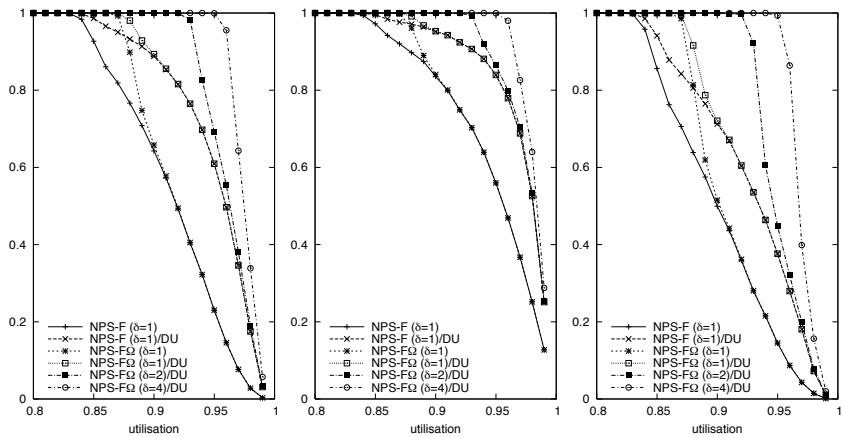

Fig. 14 Fraction of schedulable task sets of a given utilisation range, over 4 processors. Task utilisations conform to the bimodal/exponential/uniform distribution (left to right).

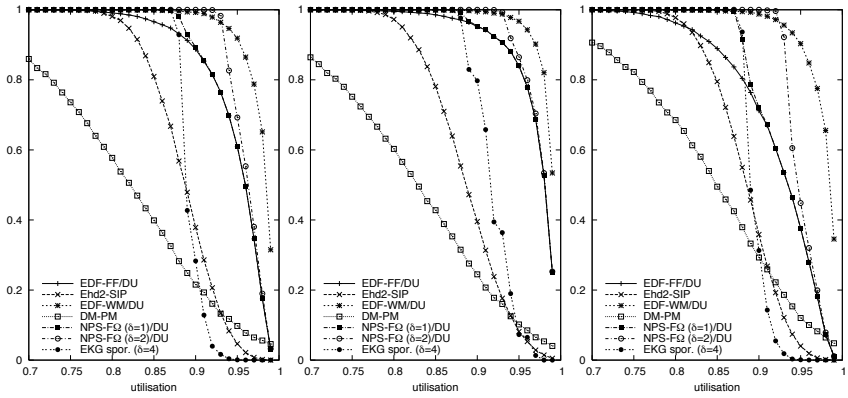

Fig. 15 Fraction of schedulable task sets of a given utilisation range, over 4 processors. Task utilisations conform to the bimodal/exponential/uniform distribution (left to right).

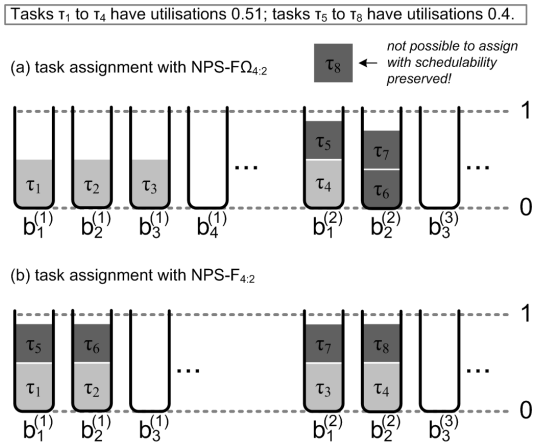

Fig. 16 A case where the Ω-optimisation is detrimental to schedulability (using clustering).

6.3 Experiments for 8 processors

The left and middle column of Figure 17 display the effect of the various configuration settings, for NPS-F and NPS-F$_{m:4}$ (clustered) respectively. (The graphs also include a variant, NPS-F$\Omega^{+}_{m:\mu}$, which we will discuss later.) For 8 processors and the non-clustered algorithm, the Ω-optimisation offers noticeable improvement, whether the task set is ordered or not.

For the clustered algorithm however, we observe a paradox: If the task set is more ordered than the default partial ordering (i.e. with /opt and /DU orderings), the clustered algorithm performs better *without* the Ω-optimisation. We confirmed that this is not an error. There exist a few task sets (more than vice versa) which are schedulable with NPS-F$_{m:4}$ but not under NPS-F$\Omega_{m:4}$. The reason is best illustrated via the next example:

Example 2 Assume 4 processors, in clusters of 2. There are 8 tasks to be scheduled: τ_1 to τ_4 have utilisation 0.51 and τ_5 to τ_8 have utilisation 0.4. Using NPS-F$\Omega_{4:2}$, it is possible to assign τ_3 to the first cluster (Figure 16(a)) with schedulability preserved, unlike when using NPS-F$_{4:2}$ (Figure 16(a)). However, hardly any remaining scheduling capacity is then left in that cluster – and the other cluster cannot accommodate all remaining tasks.

Using NPS-F$\Omega_{4:2}$, τ_3 is placed in a newly-opened bin. In the end, the bins of the first-cluster are "half-empty", thus their contents require more inflation overall, than if they could have fitted in fewer (hence fuller) bins. But without the Ω-optimisation, τ_3 ends up on the second cluster and the lower-utilisation tasks towards the end smooth out the fragmentation in both clusters.

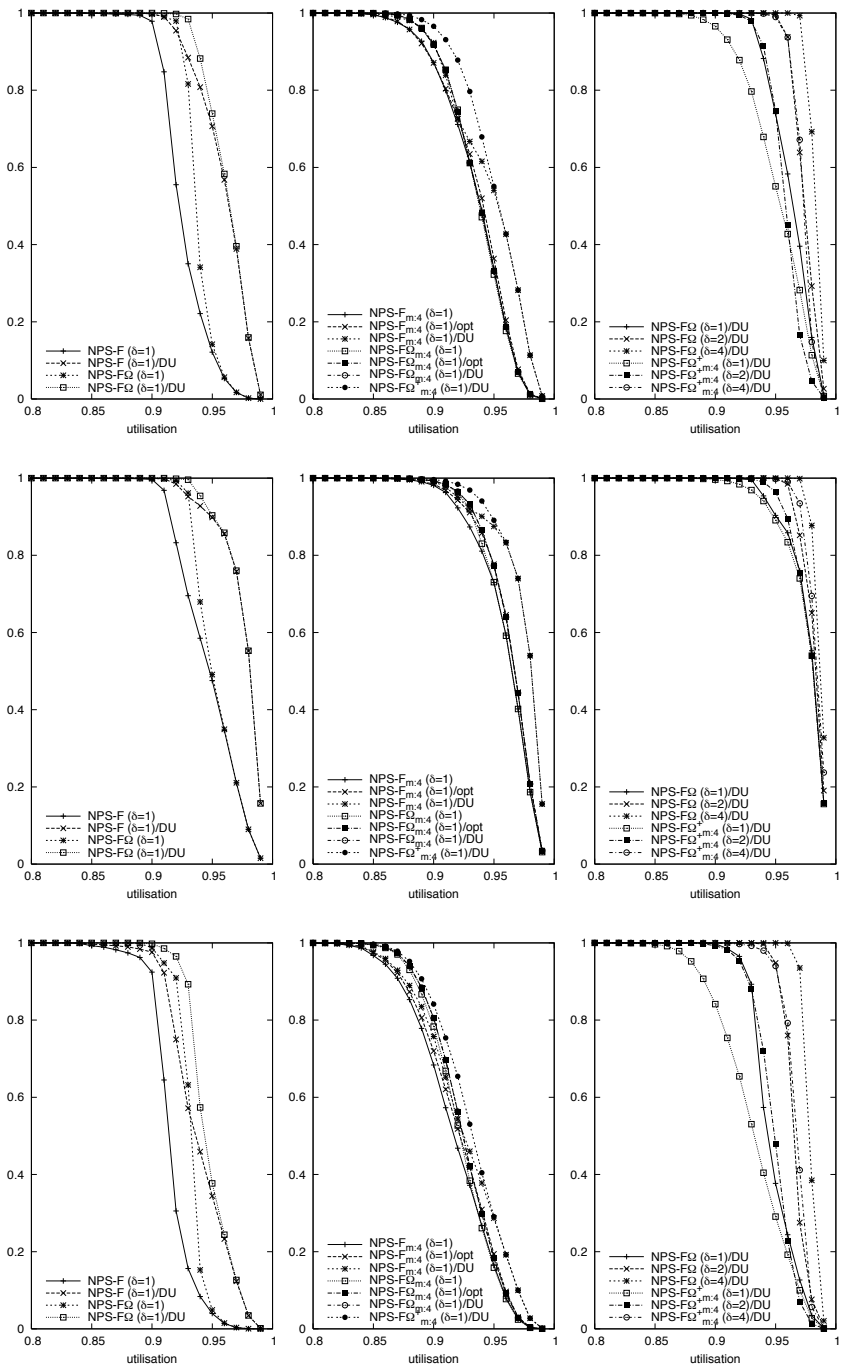

Fig. 17 Fraction of schedulable task sets of a given utilisation range, over 8 processors. Task utilisations follow the bimodal/exponential/uniform distribution (top to bottom row).

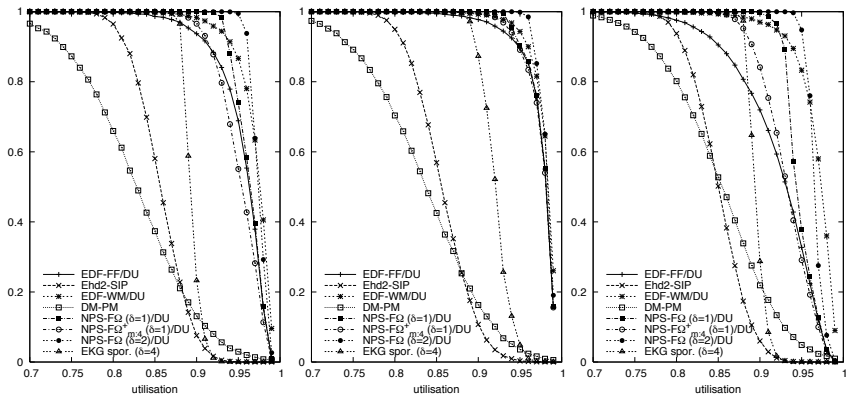

Fig. 18 Fraction of schedulable task sets of a given utilisation range, over 8 processors. Task utilisations follow the bimodal/exponential/uniform distribution (left to right).

This behavior can be remedied by using the non-Ω-optimised schedulability condition for assignments on a cluster up until a task cannot be assigned anywhere with schedulability preserved; then switching to Ω-optimised schedulability test and repeating assignment attempts for the task in consideration. Assignment attempts for any remaining tasks, would also use the Ω-optimised schedulability test. This scheme (designated as NPS-F$\Omega^{+}_{m:\mu}$) would strictly dominate the non-Ω-optimised clustered algorithm. Also, remaining tasks at that point would have lower utilisations (due to the task ordering), which would make it likelier for them to be eventually assigned with schedulability preserved. The graphs for NPS-F$\Omega^{+}_{m:\mu}$ in Figure 17 demonstrate the considerable resulting improvement. Finally, the right column of Figure 17 presents the noticeable effect of δ on schedulability. Overall, NPS-F$\Omega^{+}_{m:\mu}$ does not perform much worse than the respective non-clustered variant.

Figure 18 compares the performance of NPS-F (clustered and non-clustered) with other algorithms. Note that even with $\delta = 1$, NPS-F performs similarly well to EDF-WM/DU. However, its performance is also more consistent; the cut-off in performance comes at higher utilisations and it is steeper. Interestingly, even though the utilisation bound for $\delta = 1$ is (only 75%) NPS-FΩ appears capable of scheduling almost all tasks at up to 90% utilisation in this experiment and most tasks at even 95% utilisation.

7 Practical considerations

Our discussion so far assumed that, other than the processors themselves, tasks share no other resources. Such simplifying assumptions, though unrealistic, are common practice when an algorithm is introduced. In our case, this facilitated focusing on the core concepts of the approach and also the derivation of its utilisation bound.

However, there is no fundamental reason why a shared resource management could not work with NPS-F. Given that NPS-F amounts, for all practical purposes, to partitioned scheduling over notional processors, any resource management protocol that applies to partitioned multiprocessors would work.

Of course, in the general case, there may exist performance issues. Consider a task executing on a fractional capacity notional processor $\tilde{P}_p$ with exclusive access to a shared resource, which is preempted because the reserve for the notional processor runs out: the task cannot resume execution until its next reserve becomes available – which may take too long. In turn, this increases blocking times for other tasks, mapped to other notional processors, which request the resource.

In principle, both of these issues could be resolved using an approach similar to that suggested by Rajkumar et al (1988): All critical sections could be executed on one particular physical processor (with the task logically "migrating" there, during the respective execution). Moreover, this processor need not necessarily be dedicated for the execution of critical sections; Rajkumar et al (1988) show how application code (i.e. other tasks) can also be assigned there, to execute in the background. An approach following these principles would be practical and easy to implement.

8 Conclusions

In this article, we discussed the multiprocessor real-time scheduling algorithm NPS-F in two variants: NPS-F "proper" (i.e. non-clustered) and NPS-F$_{m:\mu}$ (i.e. clustered). Both variants aim for high schedulable utilisation with as few preemptions as possible. NPS-F, is configurable for a utilisation bound between 75% and 100%, via a parameter δ. For each configuration, its worst-case preemption counts are lower than those of any other known scheme matching the respective utilisation bound. Using the bound on preemptions vs the utilisation bound as a metric, it is the most preemption-light multiprocessor scheduling scheme, with UB $> 50\%$.

Moreover, although some preemptions under NPS-F may be costly migrations, technological advances, in the form of multicores with shared caches, offer a way of mitigating this issue. NPS-F$_{m:\mu}$, which can be described as "per chip" (rather than "per processor") partitioning, eliminates migrations across chip boundaries (which are the costliest). The moderate decrease in the utilisation bound, relative to NPS-F, is less pronounced the greater the cluster size – and cores per chip are bound to increase, in turn permitting greater cluster sizes. NPS-F$_{m:\mu}$ is a thus a *scalable* scheduling approach.

The Ω-optimisation, introduced in this paper, reduces processing capacity requirements for schedulability, thus permiting previously unschedulable tasks to be scheduled. It can be applied to both non-clustered and the clustered NPS-F. In the latter case, we had to identify an apparent performance anomaly and fix this via a heuristic, so as to obtain the same degree of improvement as with the non-clustered variant.

Finally, we also evaluated the scheduling potential of our algorithm via experiments. The results showed that NPS-F was capable of scheduling almost all tasks even for utilisations ranging well above its utilisation bound. Even with the most preemption-light configuration ($\delta = 1$) with UB=75%, over 8 processors, almost all tasks sets with utilisation up to 90% were schedulable. Thus NPS-F performs even better in practice than its utilisation bound suggests.

Acknowledgements

This work was partially funded by the Portuguese Science and Technology Foundation (Fundação para a Ciência e a Tecnologia – FCT) and the European Commission through grant ArtistDesign ICT-NoE-214373 and the Luso-American Development Foundation (FLAD).

References

AMD Inc (2008a) Key Architectural Features of AMD Phenom X3 Triple-Core Processors. Product information – http://www.amd.com/us-en/Processors/ProductInformation/0,,30_118_15331_15332%5E15615,00.html

AMD Inc (2008b) Key Architectural Features of AMD Phenom X4 Quad-Core Processors. Product information – http://www.amd.com/us-en/Processors/ProductInformation/0,,30_118_15331_15332%5E15334,00.html

AMD Inc (2008c) Quad-Core AMD Opteron Processor. Product brief – http://www.amd.com/us-en/Processors/ProductInformation/0,,30_118_8796_15223,00.html

Anderson J, Srinivasan A (2004) Mixed Pfair/ERfair scheduling of asynchronous periodic tasks. Journal of Computer and System Sciences 68(1):157–204

Anderson JH, Bud V, Devi UC (2005) An EDF-based Scheduling Algorithm for Multiprocessor Soft Real-Time Systems. In: Proceedings of the 17th Euromicro Conference on Real-Time Systems, pp 199–208

Anderson JH, Calandrino JM, Devi UC (2006) Real-Time Scheduling on Multicore Platforms. In: Proceedings of 12th IEEE Real-Time and Embedded Technology and Applications Symposium, pp 179–190

Andersson B, Bletsas K (2008) Sporadic Multiprocessor Scheduling with Few Preemptions. In: Proceedings of the 20th Euromicro Conference on Real-Time Systems (ECRTS), pp 243–252

Andersson B, Bletsas K, Baruah S (2008) Scheduling arbitrary-deadline sporadic task systems on multiprocessors. In: Proc. of 29th Real-Time Systems Symposium (RTSS), pp 385–394

ARM Ltd (2008) ARM11 MPCore. Product information – available online at http://www.arm.com/products/CPUs/ARM11MPCoreMultiprocessor.html

Baruah SK, Mok AK, Rosier LE (1990) Preemptively scheduling hard-real-time sporadic tasks on one processor. In: Proceedings of the 11th IEEE Real-Time Systems Symposium, pp 182–190

Baruah SK, Cohen NK, Plaxton CG, Varvel DA (1996) Proportionate progress: A notion of fairness in resource allocation. Algorithmica 15(6):600–625

Bletsas K, Andersson B (2009a) Notional processors: an approach for multiprocessor scheduling. In: Proceedings of the 15th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), pp 3–12

Bletsas K, Andersson B (2009b) Preemption-light multiprocessor scheduling of sporadic tasks with high utilisation bound. In: Proc. of 30th Real-Time Systems Symposium (RTSS), pp 447–456

Brandenburg BB, Calandrino JM, Anderson JH (2008) On the scalability of real-time scheduling algorithms on multicore platforms: A case study. In: Proc. of 29th Real-Time Systems Symposium (RTSS), pp 157–169

Calandrino JM, Anderson JH, Baumberger DP (2007) A Hybrid Real-Time Scheduling Approach for Large-Scale Multicore Platforms. In: Proceedings of 19th Euromicro Conference on Real-Time Systems, pp 247–258

Carpenter J, Funk S, Holman P, Anderson J, Baruah S (2004) Handbook on Scheduling Algorithms, Methods and Models, Chapman Hall/CRC, Boca, chap 30: A Categorization of Real-time Multiprocessor Scheduling Problems and Algorithms

Chao Y, Lin S, Lin K (2008) Schedulability issues for EDZL scheduling on real-time multiprocessor systems. Information Processing Letters 107(5):158–164

Cho S, Lee S, Han A, Lin K (2002) Efficient Real-Time Scheduling Algorithms for Multiprocessor Systems. IEICE Trans on Communications E85-B(12):2859–2867

Devi U, Anderson J (2005) Tardiness bounds for global EDF scheduling on a multiprocessor. In: Proceedings of the 26th IEEE Real-Time Systems Symposium, pp 330–341

Fedorova A, Seltzer M, Small C, Nussbaum D (2005) Performance of multithreaded chip multiprocessors and implications for operating system design. In: Proceedings of the USENIX 2005 Annual Technical Conference

Fisher N, Goossens J, Baruah S (2010) Optimal online multiprocessor scheduling of sporadic real-time tasks is impossible. Real-Time Systems 45:26–71

Garey MR, Johnson DS (1979) Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman & Co

Intel Corporation (2008a) Intel Core i7 Processor. Product brief – http://download.intel.com/products/processor/corei7/prod_brief.pdf

Intel Corporation (2008b) Intel Xeon Processor 3500 Series. http://www.intel.com/cd/channel/reseller/asmo-na/eng/products/server/processors/3500/feature/index.htm

Intel Corporation (2008c) Intel Xeon Processor 7400 Series. Datasheet – http://download.intel.com/design/xeon/datashts/32033501.pdf

Intel Corporation (2009) Intel Xeon Processor 5500 Series. Product brief – http://download.intel.com/products/processor/xeon/dc55kprodbrief.pdf

Kato S, Yamasaki N (2007) Real-Time Scheduling with Task Splitting on Multiprocessors. In: Proc. of the 13th IEEE International Conf. on Embedded and Real-Time Computing Systems and Applications (RTCSA), pp 441–450
Kato S, Yamasaki N (2008) Portioned static-priority scheduling on multiprocessors. In: Proceedings of the IEEE International Parallel and Distributed Processing Symposium (IPDPS), pp 1–12
Kato S, Yamasaki N (2009) Semi-partitioned fixed-priority scheduling on multiprocessors. In: Proceedings of the 15th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), pp 23–32
Kato S, Yamasaki N, Ishikawa Y (2009) Semi-partitioned scheduling of sporadic task systems on multiprocessors. In: Proceedings of the 21st Euromicro Conference on Real-Time Systems (ECRTS), pp 249–258
Leung J, Whitehead J (1982) On the complexity of fixed-priority scheduling of periodic real-time tasks. Performance Evaluation 2(4):237–250
Mok AK (1983) Fundamental design problems of distributed systems for the hard real-time environment. PhD thesis, MIT
Rajkumar R, Sha L, Lehoczky JP (1988) Real-time synchronization protocols for multiprocessors. In: Proceedings of the 9th IEEE Real-Time Systems Symposium (RTSS 1988), pp 259–269
Shin I, Easwaran A, Lee I (2007) Hierarchical Scheduling Framework for Virtual Clustering of Multiprocessors. In: Proceedings of the 20th Euromicro Conference on Real-Time Systems, pp 181–190

Appendix

Theorem 5 *A periodic reserve can always accommodate implicit-deadline tasks of cumulative utilisation $U \leq 1$, scheduled under EDF, if it measures $\frac{(\delta+1)\cdot U}{U+\delta} \cdot S$, provided that the timeslot length S does not exceed $\frac{1}{\delta}$ times the interarrival time of any task served.*

Proof Assume a deadline miss (the earliest one by a task served by the reserve) at $t{=}t_m$. Then, let $t_m - L$ denote the earliest time before t_m such that, all sub-intervals of $[t_m - L,\ t_m)$ which lie within the periodic reserve, will have been busy. Then, let t_d denote the cumulative execution requirement, over $[t_m - L,\ t_m)$, of all jobs by tasks served by the reserve which arrived at $t = t_m - L$ or later and whose deadlines lie no later than t_m. Additionally, let t^{φ} denote the cumulative time available to tasks served by the reserve (i.e. the time lying inside the reserve). The missed deadline at t_m means:

$$t_d > t^{\varphi} \tag{35}$$

Regarding t_d, it follows from (Baruah et al, 1990) that

$$t_d \leq \sum_{\tau_i \in \tau} \left\lfloor \frac{L}{T_i} \right\rfloor \cdot C_i \overset{(35)}{\Longrightarrow} \sum_{\tau_i \in \tau} \left\lfloor \frac{L}{T_i} \right\rfloor \cdot C_i > t^{\varphi} \tag{36}$$

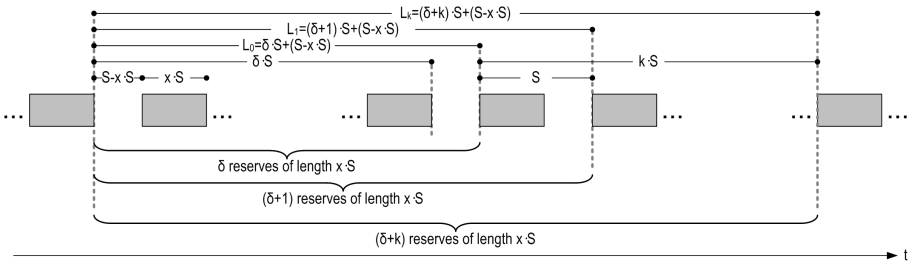

Fig. 19 Of all time windows of length $\delta \cdot S$ or more, which start as a reserve ends and end as a reserve starts, L_0 is the one for which the ratio of time t^{φ} belonging to the reserves to the overall interval length, is minimal. This stems from the fact that, $\forall k \geq 1$: $\frac{\delta \cdot x \cdot S}{\delta \cdot S + (S - x \cdot S)} \leq \frac{(\delta+k) \cdot x \cdot S}{(\delta+k) \cdot S + (S - x \cdot S)}$

At this point we note that

$$\sum_{\tau_i \in \tau} \left\lfloor \frac{L}{T_i} \right\rfloor \cdot C_i \leq \sum_{\tau_i \in \tau} \left(\frac{L}{T_i} \cdot C_i \right) = L \cdot \sum_{\tau_i \in \tau} \frac{C_i}{T_i} = L \cdot U \Rightarrow$$

$$\overset{(36)}{\Longrightarrow} L \cdot U > t^{\varphi} \Rightarrow U > \frac{t^{\varphi}}{L} \tag{37}$$

Inequality 37 states that as long as, within any interval of length $L \geq S$, it holds that t^{φ} (i.e. the time available for the execution of tasks served by the reserve), as a fraction of L (i.e. the interval length), is no less than U, then deadlines by tasks served by the reserve will always be met. Thus, for deadlines to always be met, a sufficient condition is:

$$U \leq \frac{t^{\varphi}}{L} \tag{38}$$

Time for the execution of tasks served by the reserve is available as periodic time windows of length $x \cdot S \leq S$ (corresponding to the reserves), interleaved by time windows of length $S - x \cdot S$ (during which, tasks served by the reserve cannot execute). Then, the most unfavorable selection of an offset, relative to reserve boundaries, as the start of an interval of a given length (in terms of time available to tasks served by the reserve, within said interval) is immediately past the end of a reserve. Then, of all time windows of length $L \geq \delta \cdot S$, the one within which, the cumulative time belonging to reserves (i.e. t^{φ}), divided by L is minimised, is the one with $L = \delta \cdot S + (S - x \cdot S)$ (because it ends just as the next reserve begins). For an intuitive illustration, see Figure 19.

In that case, $t^{\varphi} = \delta \cdot x \cdot S$ and

$$\frac{t^{\varphi}}{L} = \frac{\delta \cdot x \cdot S}{\delta \cdot S + (S - x \cdot S)} = \frac{\delta \cdot x}{\delta + 1 - x} \overset{(38)}{\Rightarrow} U \leq \frac{\delta \cdot x}{\delta + 1 - x} \Rightarrow x \geq \frac{(\delta + 1) \cdot U}{U + \delta}$$

which proves the theorem. □

Theorem 6 *For a given value of δ, the utilisation bound of NPS-F is greater than that of the algorithm in (Andersson and Bletsas, 2008).*

Proof It suffices to show that $\mathrm{UB}_{\text{NPS-F}} - \mathrm{UB}_{\ 2008} > 0\ \forall\delta$ (where $\mathrm{UB}_{\text{NPS-F}}$, $\mathrm{UB}_{\ 2008}$ denote the respective utilisation bounds, which are functions of δ). By inspection,

$$\mathrm{UB}_{\text{NPS-F}} = \frac{2\cdot\delta+1}{2\cdot\delta+2} > \frac{2\cdot\delta}{2\cdot\delta+1} = 1-2\cdot\alpha(\tfrac{1}{2}) \tag{39}$$

and

$$\mathrm{UB}_{\ 2008} = 4\cdot(\sqrt{\delta\cdot(\delta+1)}-\delta)+1 = 1-2\cdot\alpha(U_0) \tag{40}$$

where $U_0 = \sqrt{\delta\cdot(\delta+1)}-\delta$. Hence

$$(39),(40) \Rightarrow \mathrm{UB}_{\text{NPS-F}} - \mathrm{UB}_{\ 2008} > 2\cdot\big(\alpha(U_0)-\alpha(\tfrac{1}{2})\big)$$

But $\alpha(U_0) > \alpha(\frac{1}{2})$, because the function $\alpha(U)$ has a maximum at $U = U_0$. Hence $\mathrm{UB}_{\text{NPS-F}} - \mathrm{UB}_{\ 2008} > 0$. □

Theorem 7 *For $\delta = 1$, if tasks are ordered such that tasks with utilisation $\frac{1}{2}$ or higher (i) are indexed in order of decreasing utilisation and (ii) precede all other tasks, it holds that* $\mathrm{UB}_{m:4} = \frac{5}{8}$.

Proof We will first show that every task set τ with $U_\tau \leq \frac{5}{8}$ is schedulable. Suppose that an unschedulable task set existed with $U_\tau \leq \frac{5}{8}$. Then, the bin-packing algorithm would encounter a task τ_f with utilisation u_f not assignable to any bin of any cluster (subject to assignments already made) and would exit declaring failure (Figure 7, line 11). (Recall that $m''^{(q)}$ denotes the index of the highest-indexed bin associated with Q_q with tasks assigned to it, immediately before attempting to assign τ_f.) Regarding τ_f, one of these mutually exclusive cases holds:

- Case 1: $\frac{5}{8} < u_f \leq 1$
  Then, due to the task ordering, all previously assigned tasks had utilisations above $u_f > \frac{5}{8}$. Also, every cluster has no fewer than $\mu = 4$ tasks assigned to it, (or else τ_f would have been assigned). Therefore every one of the $\frac{m}{4}$ clusters has tasks assigned to it of cumulative utilisation above $4\cdot\frac{5}{8} = \frac{5}{2}$.
- Case 2: $\frac{1}{2} < u_f \leq \frac{5}{8}$
  Then, due to the task ordering, all previously assigned tasks had utilisations above $u_f > \frac{1}{2}$. This also means that each assigned task is the single task assigned to its bin. Thus, in every cluster Q_q, bins $b_1^{(q)}$ to $b_{m''^{(q)}}^{(q)}$ are all utilised above $\frac{1}{2}$. Also, from Inequality 16, it holds for every cluster Q_q that $m''^{(q)}>3$ (or the assignment of τ_f would not have failed). Thus, for each cluster Q_q, two complementary possibilities remain:
  - Case 2a: $m''^{(q)} = 4$
    The assignment of τ_f failed, thus it could not be assigned neither to some bin among $b_1^{(q)}$ to $b_{m''^{(q)}}^{(q)}$ (together with other tasks) nor to

$b^{(q)}_{m''^{(q)}+1}$ (on its own), while also satisfying Inequality 16. In particular, from the failed assignment attempt on $b^{(q)}_{m''^{(q)}+1}$, we deduce from Inequality 16 that

$$\text{inflate}(u_f) + \sum_{p=1}^{m''^{(q)}} \text{inflate}(U_p^{(q)}) > \mu \overset{m''^{(q)}=4}{\overset{\mu=4}{\Longrightarrow}}$$

$$\sum_{p=1}^{4} \text{inflate}(U_p^{(q)}) > 4 - \text{inflate}(u_f) \geq 4 - \text{inflate}\left(\frac{5}{8}\right) = \frac{42}{13} \overset{Th.\,2}{\Rightarrow}$$

$$4 \cdot \text{inflate}(\bar{U}^{(q)}) > \frac{42}{13} \Rightarrow \text{inflate}(\bar{U}^{(q)}) > \frac{21}{26} \Rightarrow \bar{U}^{(q)} > \frac{21}{31}$$

But then, Q_q has tasks of cumulative utilisation above $4 \cdot \frac{21}{31} = \frac{84}{31} > \frac{5}{2}$ already assigned to it prior to the attempt to assign τ_f.

- Case 2b: $m''^{(q)} \geq 5$
  Then, bins $b_1^{(q)}$ to $b_5^{(q)}$, each have a task assigned to them, prior to the attempted assignment of τ_f. Due to the task ordering, these tasks all have utilisations no less than u_f (which in turn exceeds $\frac{1}{2}$, as per the assumption of Case 2b). Thus, Q_q has tasks already assigned to it of cumulative utilisation above $5 \cdot \frac{1}{2} = \frac{5}{2}$, before attempting to assign τ_f.

In either Case 2a/b, the cumulative utilisation of tasks assigned to Q_q exceeds $\frac{5}{2}$ before trying to assign τ_f.

- Case 3: $0 < u_f \leq \frac{1}{2}$
  The utilisation bound of (non-clustered) NPS-F is $\frac{3}{4}$. Therefore, on a 4-processor cluster, tasks of cumulative utilisation up to $\frac{3}{4}\cdot 4{=}3$ are always schedulable under NPS-F. Hence, if the cumulative utilisation of tasks already assigned to some cluster Q_q before attempting to assign τ_f did not exceed $\frac{5}{2}$, then it would have been possible to assign τ_f (of utilisation $u_f < \frac{1}{2}$) to Q_q. But τ_f could not be assigned, subject to previous assignments, hence, on every cluster, the cumulative utilisation of tasks already assigned before the attempt to assign τ_f exceeds $\frac{5}{2}$.

In any case, if some task cannot be assigned, subject to prior assignments, then every one of the $\frac{m}{4}$ clusters already has tasks assigned to it of cumulative utilisation above $\frac{5}{2}$, before attempting to assign τ_f. This would mean that the cumulative utilisation of tasks assigned to any of the $\frac{m}{4}$ clusters (a subset of τ) exceeds $\frac{m}{4} \cdot \frac{5}{2} = \frac{5}{8} \cdot m$. Therefore, τ cannot be unschedulable unless $U_\tau > \frac{5}{8}$ – which contradicts the assumption that $U_\tau \leq \frac{5}{8}$. Therefore $\text{UB}_{m:4} \not< \frac{5}{8}$. To show that, in fact, $\text{UB}_{m:4} = \frac{5}{8}$, it suffices to find an unschedulable task set with U_τ arbitrarily close to $\frac{5}{8}$. This is the case for a set of $5 \cdot k + 1$ tasks, each of utilisation $\frac{1}{2} + \epsilon$, if $m = 4 \cdot k$ and $k \to \infty$ and $\epsilon \to 0^+$. □

Theorem 8 *It suffices, for a set τ' of implicit-deadline tasks with cumulative utilisation U to be schedulable under EDF, if, within any possible time interval of length $L \geq \min_{\tau_i \in \tau'} T_i$, the time t^φ available for the execution of τ' (possibly in disjoint time intervals) is $U \cdot L$ or greater.*

Proof Assume a deadline miss (the earliest one to occur) at $t = t_m$. Then, let $t_m - \Lambda$ denote the earliest time before t_m such that, during all sub-intervals of $[t_m - \Lambda,\ t_m)$ available for the execution of τ', some task from τ' is executing. Then, let t_d denote the cumulative execution requirement, over $[t_m - \Lambda,\ t_m)$, of all jobs by τ' which arrived at $t = t_m - L$ or later and whose deadlines lie no later than t_m. The missed deadline at t_m means:

$$t_d > t^{\varphi} \tag{41}$$

Regarding t_d, it follows from (Baruah et al, 1990) that

$$t_d \leq \sum_{\tau_i \in \tau'} \left\lfloor \frac{\Lambda}{T_i} \right\rfloor \cdot C_i \overset{(41)}{\Longrightarrow} \sum_{\tau_i \in \tau'} \left\lfloor \frac{\Lambda}{T_i} \right\rfloor \cdot C_i > t^{\varphi} \tag{42}$$

At this point we note that

$$\sum_{\tau_i \in \tau'} \left\lfloor \frac{\Lambda}{T_i} \right\rfloor \cdot C_i \leq \sum_{\tau_i \in \tau'} \left(\frac{\Lambda}{T_i} \cdot C_i \right) = \Lambda \cdot \sum_{\tau_i \in \tau'} \frac{C_i}{T_i} = \Lambda \cdot U \overset{(42)}{\Longrightarrow} \Lambda \cdot U > t^{\varphi}$$

$$\Rightarrow U > \frac{t^{\varphi}}{\Lambda} \tag{43}$$

If then it is somehow ensured that, for any possible interval of length $L \geq \min_{\tau_i \in \tau'} T_i$, it holds that $\frac{t^{\varphi}}{L} \geq U$, this condition suffices for the schedulability of τ' under EDF. □