Technical Report

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

Konstantinos Bletsas

Björn Andersson

HURRAY-TR-090907

Version: 0

Date: 09-11-2009

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 a scheduling scheme with as few preemptions and migrations as possible, for a given guaranteed utilisation bound, we respond with a new 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 it in terms of its utilisation bound.

We also introduce a clustered variant of the algorithm, for use with systems made of multicore chips. It eliminates off-chip task migrations, which are costly, by dividing processors into independently-scheduled clusters (each, using the non-clustered algorithm). Each cluster is formed out of cores on the same chip. (The cluster size is a parameter to the algorithm.) We show that the utilisation bound is only moderately affected.

© IPP Hurray! Research Group
www.hurray.isep.ipp.pt

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

Konstantinos Bletsas and Björn Andersson
IPP-HURRAY Research Group, CISTER/ISEP, Polytechnic Institute of Porto,
Rua Dr. António Bernardino de Almeida 431, 4200-072 Porto, Portugal
ksbs@isep.ipp.pt, bandersson@dei.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 a scheduling scheme with as few preemptions and migrations as possible, for a given guaranteed utilisation bound, we respond with a new 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 it in terms of its utilisation bound.

We also introduce a clustered variant of the algorithm, for use with systems made of multicore chips. It eliminates off-chip task migrations, which are costly, by dividing processors into independently-scheduled clusters (each, using the non-clustered algorithm). Each cluster is formed out of cores on the same chip. (The cluster size is a parameter to the algorithm.) We show that the utilisation bound is only moderately affected.

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 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 $0 \leq C_i \leq T_i$). 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 $U_\tau \leq \mathrm{UB}$ 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% [14]. Conversely, the *pfair* family of global scheduling algorithms offers utilisation bounds of 100% [9][4] but at the cost of numerous preemptions [17].

Hybrid approaches aim for combination of strengths: EDF-fm [5] schedules soft, not hard, real-time tasks at up to 100% system utilisation with limited tardiness. Ehd2-SIP [23] and EDDP [24], with utilisation bounds of 50% and 65%, typically generate few preemptions, although no respective upper bound is known. Under EKG-sporadic [7] 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. Currently, the most "preemption-light" (in terms of a proven upper bound on preemptions) of all schemes with utilisation bounds above 50% is Notional Processor Scheduling (NPS) [11], however its utilisation bound is just $66.\bar{6}\%$. Note that although EDZL [16] has even fewer preemptions, it is yet unproven whether its utilisation bound exceeds 50% [15].

We introduce a new algorithm NPS-F, 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 [18][6].

NPS-F stands for "Notional Processor Scheduling – Fractional capacity". It is related to NPS [11] 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 concludes.

2. Useful concepts

2.1. On periodic reserves

A *periodic reserve* is a kind of server, for scheduling one or more sporadic tasks. 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.

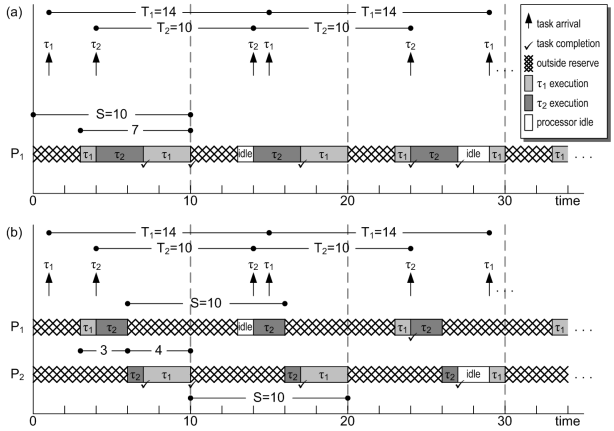

Figure 1. Scheduling multiple tasks within (a) a single periodic reserve (b) temporally adjacent reserves on different processors. Observe that in example (b), wherein the two reserves summed match in duration the single reserve in the first example (with identical arrivals and timeslot length), task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

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 [11], 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 $0{=}a_0{<}a_1{<}\ldots{<}a_z{\leq}1$, $h_u{\in}\{P_1,P_2,\ldots,P_m\}$, $\forall u{\in}\{1,2,\ldots,z-1\}$ and $0{\leq}\omega{<}1$. 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 $\text{deflate}(a_z)$.

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

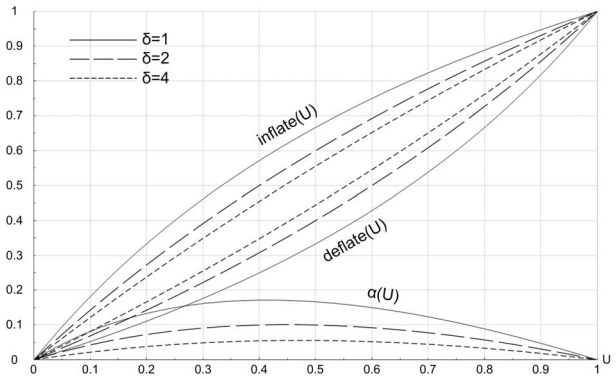

Figure 2. Functions inflate, deflate and α, for different values of parameter δ.

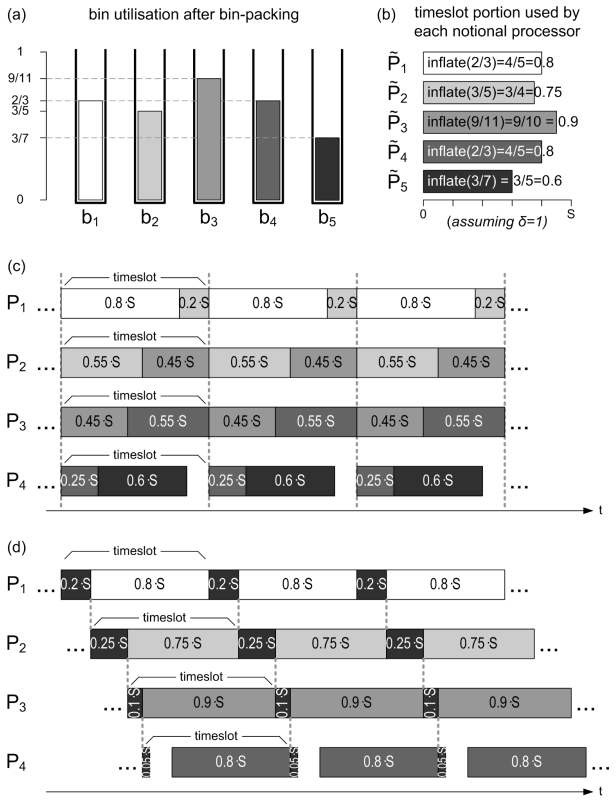

Figure 3. Alternative approaches to mapping notional to physical processors.

3. The new algorithm NPS-F

Consider 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 $\text{inflate}(U_p)$. In turn, all m'' notional processors are implemented upon the m physical processors (P_1 to P_m).

In an example, Figure 3(a) depicts bin utilisations after bin-packing and Figure 3(b) shows the processing capacity requirements for corresponding notional processors.

Two alternatives (equivalent, in terms of scheduling potential) exist for mapping notional processors to physical ones. Under *flat* mapping (Figure 3(c)), each notional processor (and thus, each task) migrates between at most two physical processors. Under *semi-partitioned* mapping (Figure 3(d)), m notional processors (and the associated tasks) always stay on one physical processor, but the rest may need to migrate between many. The two approaches are described in pseudocode in Figure 4. A mapping is feasible (whether flat or semi-partitioned) if and only if:

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

3.1. Utilisation bound

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*

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

Proof: Let τ_f denote the first task, during the bin-packing procedure, to be assigned to bin $b_{m''}$ and let u_f denote the utilisation of τ_f. It then holds that

$$u_f + \sum_{p=1}^{m''-1} U_p > \frac{(m''-1)+1}{2} = \frac{m''}{2}$$

or else τ_f would have been assigned to one of bins b_1 to $b_{m''-1}$ (from Theorem 4 in [25], applied with parameter $\beta = 1$ – most pessimistically). Thus, after assigning τ_f

```
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]);
27.       ω[np][0]:=fractional_part_of(tmp);
28.       last_ω:=ω[np][0];
29.     end for
30.     last_ω:=0; //rewind;
31.     spent:=0; //rewind;
32.     for np:=m+1 to m" do
33.       a[np][0]:=0;
34.       h[np][0]:=last_h;
35.       z:=1;
36.       acc:=0;
37.       ω[np]:=last_ω; //initialise
38.       while (acc<U[np]) do
39.         if (acc+1-inflate(U[last_h])-spent<U[np]) then
40.           a[np][z]:=acc+(1-inflate(U[last_h]))-spent;
41.           h[np][z]:=last_h;
42.           acc:=a[np][z];
43.           z:=z+1;
44.           last_h:=last_h+1;
45.           spent:=0;
46.         else
47.           a[np][z]:=U[np];
48.           h[np][z]=last_h;
49.           last_ω:=(last_ω+U[np]) modulo 1;
50.           spent:=a[np][z]-a[np][z-1];
51.         end if
52.       end while
53.     end for
54.   end if
55. end procedure
```

Figure 4. Notional processor implementation.

and even before assigning any other yet unassigned tasks (which cannot decrease any U_p), it holds that

$$\sum_{p=1}^{m''} U_p \geq u_f + \sum_{p=1}^{m''-1} U_p > \frac{m''}{2} \Rightarrow \frac{\sum_{p=1}^{m''} U_p}{m''} > \frac{1}{2}$$

□

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

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,

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

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

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

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

□

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:

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

□

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:

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

Therefore, for schedulability, it suffices that

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

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

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

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

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

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

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. 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:

- type-α_1: Occurring when the reserve implementing a notional processor runs out. One such preemption per notional processor per timeslot occurs. It is potentially a migration.
- type-β_1: Occurring whenever a notional processor migrates to another physical processor. One such preemption per physical processor per timeslot occurs. It is always a migration.

With semi-partitioned mapping:

- type-α_2: Occurring when the reserve implementing a notional processor runs out. One such preemption per notional processor per timeslot occurs. For notional processors $\tilde{P}_1$ to $\tilde{P}_m$ it is never a migration.
- type-β_2: Occurring whenever a notional processor migrates to another physical processor. One such preemption per physical processor per timeslot occurs. It is always a migration.

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

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

$$= N_{arr}(\Delta t) + \left\lceil \frac{\Delta t}{S} \right\rceil \cdot (m + m'') \quad (13)$$

where $N_{arr}(\Delta t)$ is an upper bound on task arrivals within the interval. We next eliminate m'' from Equation 13:

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

UB	$\delta = 1$	$\delta = 2$	$\delta = 3$	$\delta = 4$	$\delta \to \infty$
NPS-F	75%	83.3%	87.5%	90%	→100%
EKG-spor. [7]	65.7%	79.8%	85.6%	88.9%	→100%

Table 1. Comparison of utilisation bounds

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 13 and Corollary 1, we obtain:

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

$$N_{pr}(\Delta t) < N_{arr}(\Delta t) + \left\lceil \frac{\Delta t}{\text{TMIN}} \right\rceil \cdot 3 \cdot m \cdot \delta \quad (14)$$

This bound for preemptions is the same as for the variant of EKG for sporadic tasks [7], 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 [7] cannot schedule *any* task set whose utilisation exceeds its utilisation bound. NPS-F thus dominates that scheme without causing more preemptions.

Yet a comparison, in terms of preemptions, of NPS-F with the original NPS [11] 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 δ=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 14). However, it also holds that, for both NPS and NPS-F(δ=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].

1. Note that the terminology differs slightly in [11]. In [11], 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.

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.

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 [8]. The major manufacturers already offer affordable dual-,triple- [1], quad-, six- [21] and eight-cores. Common to most latest designs is a top-level cache shared by all cores [20][21][19][22][3][2][1].

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 [18][6]. 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 [12][26][13]. Calandrino et al. [13] 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 [13] is just 50%. Shin et al. [26] aim to improve schedulability but allow for overlapping clusters (which runs counter to our motivation).

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 5 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:

$$\sum_{p=1}^{m''^{(q)}} \text{inflate}(U_p^{(q)}) \leq \mu \tag{15}$$

If a task can be assigned First-Fit to (one of) the bins of the cluster in consideration while satisfying Inequality 15, 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 15*, 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 $\text{UB}_{m:\mu}$,

```
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. 15 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
```

Figure 5. Bin-packing over clusters

we denote the utilisation bound of NPS-F$_{m:\mu}$. In notation, we can also use fixed values in place of m and μ.

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%	→94.1%
UB$_{m:m}$	75%	83.3%	87.5%	90%	→100%
UB$_{[7]}$	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 [7]

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 in [7] (also suffering from off-chip task migrations).

At present, we view μ=4 as the most relevant cluster size (given current chip offerings). This motivates one optimisation to the algorithm, which raises its utilisation bound for δ=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

$$N^q_{pr}(\Delta t) = N^q_{arr} + \left\lceil \frac{\Delta t}{\mathrm{TMIN}} \right\rceil \cdot 3 \cdot \mu \cdot \delta \qquad (16)$$

from Equation 14 (via substitution of m by μ). Over the entire system, preemptions are thus bounded by

$$N^{m:\mu}_{pr} = \sum_{q=1}^{\frac{m}{\mu}} N^q_{pr}(\Delta t) =$$
$$\sum_{q=1}^{\frac{m}{\mu}} N^q_{arr}(\Delta t) + \sum_{q=1}^{\frac{m}{\mu}} \left(\left\lceil \frac{\Delta t}{\mathrm{TMIN}} \right\rceil \cdot 3 \cdot \mu \cdot \delta \right)$$
$$= N_{arr}(\Delta t) + \left\lceil \frac{\Delta t}{\mathrm{TMIN}} \right\rceil \cdot 3 \cdot \mu \cdot \delta \cdot \frac{m}{\mu}$$
$$= N_{arr}(\Delta t) + \left\lceil \frac{\Delta t}{\mathrm{TMIN}} \right\rceil 3 \cdot m \cdot \delta \qquad (17)$$

This bound is the same as that for non-clustered NPS-F (Equation 14) 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

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

instead of $\frac{1}{\delta} \cdot \mathrm{TMIN}$. This optimisation has no downside and will reduce preemptions in most cases. In fact, it cures another weakness of NPS-F, also present in [7]: 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. Conclusion

We introduced a new multiprocessor real-time scheduling scheme 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. Using this metric, NPS-F "dethrones" the algorithm in [7] and offers a utilisation bound of 75% even at its most preemption-light setting. However, some preemptions under NPS-F may be costly migrations. Still, technological advances, in the form of multicores with shared caches, offer a way of mitigating this. 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.

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.

References

[1] AMD Inc. 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, 2008.

[2] AMD Inc. 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, 2008.

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

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

[5] J. H. Anderson, V. Bud, and U. C. Devi. An EDF-based Scheduling Algorithm for Multiprocessor Soft Real-Time Systems. In *Proceedings of the 17th Euromicro Conference on Real-Time Systems*, pages 199–208, 2005.

[6] J. H. Anderson, J. M. Calandrino, and U. C. Devi. Real-Time Scheduling on Multicore Platforms. In *Proceedings of 12th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 179–190, 2006.

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

[8] ARM Ltd. ARM11 MPCore. Product information – available online at http://www.arm.com/products/CPUs/ARM11MPCoreMultiprocessor.html.

[9] S. K. Baruah, N. K. Cohen, C. G. Plaxton, and D. A. Varvel. Proportionate progress: A notion of fairness in resource allocation. *Algorithmica*, 15(6):600–625, June 1996.

[10] S. K. Baruah, A. K. Mok, and L. E. Rosier. Preemptively scheduling hard-real-time sporadic tasks on one processor. In *Proceedings of the 11th IEEE Real-Time Systems Symposium*, pages 182–190, 1990.

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

[12] B. B. Brandenburg, J. M. Calandrino, and J. H. Anderson. On the scalability of real-time scheduling algorithms on multicore platforms: A case study. In *Proc. of 29th Real-Time Systems Symposium (RTSS)*, pages 157–169, 2008.

[13] J. M. Calandrino, J. H. Anderson, and D. P. Baumberger. A Hybrid Real-Time Scheduling Approach for Large-Scale Multicore Platforms. In *Proceedings of 19th Euromicro Conference on Real-Time Systems*, pages 247–258, 2007.

[14] J. Carpenter, S. Funk, P. Holman, J. Anderson, and S. Baruah. *A Categorization of Real-time Multiprocessor Scheduling Problems and Algorithms*, chapter 30, pages 30–1–30–17. Chapman Hall/CRC, Boca, 2004.

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

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

[17] U. Devi and J. Anderson. Tardiness bounds for global EDF scheduling on a multiprocessor. In *Proceedings of the 26th IEEE Real-Time Systems Symposium*, pages 330–341, 2005.

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

[19] Intel Corporation. Intel Xeon Processor 3500 Series. http://www.intel.com/cd/channel/reseller/asmo-na/eng/products/server/processors/3500/feature/index.htm.

[20] Intel Corporation. Intel Core i7 Processor. Product brief – available online at http://download.intel.com/products/processor/corei7/prod_brief.pdf, 2008.

[21] Intel Corporation. Intel Xeon Processor 7400 Series. Datasheet – available online at http://download.intel.com/design/xeon/datashts/32033501.pdf, October 2008.

[22] Intel Corporation. Intel Xeon Processor 5500 Series. Product brief – http://download.intel.com/products/processor/xeon/dc55kprodbrief.pdf, 2009.

[23] S. Kato and N. Yamasaki. 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)*, pages 441–450, 2007.

[24] S. Kato and N. Yamasaki. Portioned static-priority scheduling on multiprocessors. In *Proceedings of the IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 1–12, 2008.

[25] J. M. López, M. Garcia, J. L. Díaz, and D. F. Garcia. Worst-Case Utilization Bound for EDF Scheduling on Real-Time Multiprocessor Systems. In *Proc. of the 12th Euromicro Conference on Real-Time Systems*, pages 25–33, 2000.

[26] I. Shin, A. Easwaran, and I. Lee. Hierarchical Scheduling Framework for Virtual Clustering of Multiprocessors. In *Proceedings of the 20th Euromicro Conference on Real-Time Systems*, pages 181–190, 2007.

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:

$$t_d > t^{\varphi} \tag{18}$$

Regarding t_d, it follows from [10] that

$$t_d \leq \sum_{\tau_i \in \tau} \left\lfloor \frac{L}{T_i} \right\rfloor \cdot C_i \overset{(18)}{\Longrightarrow} \sum_{\tau_i \in \tau} \left\lfloor \frac{L}{T_i} \right\rfloor \cdot C_i > t^{\varphi} \tag{19}$$

At this point we note that

$$\sum_{\tau_i \in \tau} \left\lfloor \frac{L}{T_i} \right\rfloor \cdot C_i \leq \sum_{\tau_i \in \tau} \left(\frac{L}{T_i} \cdot C_i \right) = L \cdot \sum_{\tau_i \in \tau} \frac{C_i}{T_i} = L \cdot U$$
$$\overset{(19)}{\Longrightarrow} L \cdot U > t^{\varphi} \Rightarrow U > \frac{t^{\varphi}}{L} \tag{20}$$

Inequality 20 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:

$$U \leq \frac{t^{\varphi}}{L} \tag{21}$$

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

$$\frac{t^{\varphi}}{L} = \frac{\delta \cdot x \cdot S}{\delta \cdot S + (S - x \cdot S)} = \frac{\delta \cdot x}{\delta + 1 - x} \overset{(21)}{\Rightarrow}$$
$$U \leq \frac{\delta \cdot x}{\delta + 1 - x} \Rightarrow x \geq \frac{(\delta + 1) \cdot U}{U + \delta} \tag{22}$$

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 [7].*

Proof: It suffices to show that $\mathrm{UB}_{\text{NPS-F}} - \mathrm{UB}_{[7]} > 0\ \forall \delta$ (where $\mathrm{UB}_{\text{NPS-F}}$, $\mathrm{UB}_{[7]}$ denote the respective utilisation bounds, which are functions of δ). By inspection,

$$\mathrm{UB}_{\text{NPS-F}} = \frac{2 \cdot \delta + 1}{2 \cdot \delta + 2} > \frac{2 \cdot \delta}{2 \cdot \delta + 1} = 1 - 2 \cdot \alpha(\frac{1}{2}) \tag{23}$$

and

$$\text{UB}_{[7]} = 4 \cdot (\sqrt{\delta \cdot (\delta+1)} - \delta) + 1 = 1 - 2 \cdot \alpha(U_0) \quad (24)$$

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

$$(23), (24) \Rightarrow \text{UB}_{\text{NPS-F}} - \text{UB}_{[7]} > 2 \cdot \left(\alpha(U_0) - \alpha(\frac{1}{2}) \right)$$

But $\alpha(U_0) > \alpha(\frac{1}{2})$, because the function $\alpha(U)$ has a maximum at $U = U_0$. Hence $\text{UB}_{\text{NPS-F}} - \text{UB}_{[7]} > 0$. □

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* $\text{UB}_{m:4} = \frac{5}{8}$.

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 5, 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 15, 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 $b_{m''^{(q)}+1}^{(q)}$ (on its own), while also satisfying Inequality 15. In particular, from the failed assignment attempt on $b_{m''^{(q)}+1}^{(q)}$, we deduce from Inequality 15 that

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

$$\sum_{p=1}^{4} \text{inflate}(U_p^{(q)}) > 4 - \text{inflate}(u_f) \geq$$

$$4 - \text{inflate}\left(\frac{5}{8}\right) = \frac{42}{13}$$

$$\overset{Th.\ 2}{\Rightarrow} 4 \cdot \text{inflate}(\bar{U}^{(q)}) > \frac{42}{13} \Rightarrow$$

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

    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 $\epsilon \to 0^+$. □