Technical Report

Notional processors: an approach for multiprocessor scheduling

To appear in the proceedings of the 15th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'09)

Konstantinos Bletsas and Björn Andersson

HURRAY-TR-090101

Version: 0

Date: 01-25-2009

Notional processors: an approach for multiprocessor scheduling

Konstantinos Bletsas and 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: nap@isep.ipp.pt

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

Abstract

Consider the problem of designing an algorithm with ahigh utilisation bound for scheduling sporadic tasks withimplicit deadlines on identical processors. A task is characterisedby its minimum interarrival time and its executiontime. Task preemption and migration is permitted. Still, lowpreemption and migration counts are desirable.We formulate an algorithm with a utilisation bound noless than 66.6%, characterised by worst-case preemptioncounts comparing favorably against the state-of-the-art.

Notional processors: an approach for multiprocessor scheduling

Konstantinos Bletsas and Björn Andersson
IPP-HURRAY! Research Group, Polytechnic Institute of Porto (ISEP-IPP),
Rua Dr. António Bernardino de Almeida 431, 4200-072 Porto, Portugal
ksbs@isep.ipp.pt, bandersson@dei.isep.ipp.pt

Abstract

Consider the problem of designing an algorithm with a high utilisation bound for scheduling sporadic tasks with implicit deadlines on identical processors. A task is characterised by its minimum interarrival time and its execution time. Task preemption and migration is permitted. Still, low preemption and migration counts are desirable.

We formulate an algorithm with a utilisation bound no less than $66.\bar{6}\%$, characterised by worst-case preemption counts comparing favorably against the state-of-the-art.

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, with arrival times not controlled by the scheduling algorithm and *a priori* unknown but occuring 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. (T_i, C_i are 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 task set utilisation is defined as $U_\tau = \frac{1}{m} \cdot \sum_{i=1}^{n} \frac{C_i}{T_i}$. The utilisation bound UB of an algorithm is a threshold such that, all task sets with $U_\tau \leq \mathrm{UB}$ scheduled by said algorithm meet their deadlines.

Multiprocessor scheduling algorithms are often categorised as *partitioned* or *global*. Under global scheduling, a single dispatch queue is shared by all processors. At any moment, the m highest-priority tasks among those runnable are executing on the m processors. In contrast, under partitioned scheduling, all tasks in a partition are assigned to the same processor and may not migrate to another processor. Multiprocessor scheduling is thus transformed to many uniprocessor scheduling problems. While this simplifies scheduling and allows reuse of many results from uniprocessor scheduling, no partitioned algorithm can have a utilisation bound above 50%. Conversely, the *pfair* family of global scheduling algorithms offers utilisation bounds of 100% [5], [1] but at the cost of numerous preemptions [9]. The global scheduling scheme EDZL [8] is not preemption-prone and usually performs well, but its utilisation bound is unknown (and less than 63.1% [7]).

EDF-fm [2] introduced limited migration to partitioned EDF but for scheduling soft, not hard, real-time tasks at 100% utilisation with limited tardiness. Ehd2-SIP [11], another algorithm with limited migration based on partitioned EDF, is characterised by few preemptions on average but its utilisation bound is just 50% (i.e. same as for partitioned EDF). Under the hybrid approach in [3], most tasks utilise a single processor while a few (at most $m-1$) utilise two – but not both simultaneously, due to the dispatching policy. This algorithm is configurable for higher utilisation bounds (up to 100%) at the cost of increased preemptions. As such, it largely solves the problem of achieving a high utilisation bound without too many preemptions.

Still, this paper introduces an algorithm with a utilisation bound of $66.\bar{6}\%$ and even lower worst-case preemption counts than [3]. Let $N_{arr}(\Delta t)$ denote an upper bound on job arrivals in the system over an interval of length Δt. The preemptions generated during that interval are at most

$$N_{arr}(\Delta t) + \left\lceil \frac{\Delta t}{\min\limits_{i \in \{1,2,\ldots,n\}} \{T_i\}} \right\rceil \cdot \left(2 \cdot m + \left\lceil \frac{m}{3} \right\rceil\right)$$

This paper is organised as follows: Section 2 discusses important concepts, prerequisite to understanding our approach, formulated in Section 3. Sections 4 and 5 discuss its performance in terms of its utilisation bound and worst-case preemption counts, respectively. Section 6 concludes.

2. Useful concepts

2.1. Intuition

Any partitioned scheme, may in the worst case leave processors partly utilised so that although the cumulative

spare capacity in the system exceeds the utilisation of some yet unassigned task, the latter cannot be assigned to any processor. Assume however that, at run-time, some mechanism would ensure that on every instant when some P_a processor ceases to be idle, at least one other processor P_b is guaranteed to be idle. We could then reclaim the idle interval on P_a to execute some additional task and then migrate that to P_b when necessary. Even better, we could use such reclaimed spare capacity for scheduling an additional set of tasks, in isolation from other tasks on each processor, as if we had an additional processor (but in fact "piggybacking" on existing ones). Note that the dynamic mapping of logical processors to physical processors is not a new concept (for example, it is used in [13]) but we use it in a novel way.

2.2. On periodic reserves

Let U denote the cumulative utilisation of all tasks assigned to some processor. We seek predictability in the occurrence of idle intervals on this (and any other) processor, so as to exploit the previously explained principle and schedule additional tasks. Such predictability may be achieved via having all tasks on a processor execute within a fixed-size periodic time window – termed a *reserve* [4].

One or more tasks with implicit deadlines and cumulative utilisation $U \leq 1$ may meet deadlines, if scheduled within a periodic reserve, provided that the length of the periodic reserve is a sufficiently large fraction of the "timeslot length" S (i.e. the fixed interval between the start of one reserve and the start of the next one). This fraction will have to exceed U to account for unfavorable phasings with respect to the arrivals of the tasks served relative to the start of the next available reserve. The amount by which this fraction exceeds U is termed *reserve inflation*. Inflation is obviously undesirable. While it can be reduced by opting for shorter timeslots, this increases the number of preemptions resulting from the implementation of the reserves.

In [3] it was shown that, if a reserve is exclusive for the execution of some implicit-deadline task with utilisation U and if the timeslot length S does not exceed the interarrival time of the task, then, if the reserve size is at least

$$\left(U + \frac{U \cdot (1-U)}{1+U}\right) \cdot S$$

then the task will always meet deadlines. In that case, the sufficient inflation (as a function of U) is given by

$$\alpha(U) = \frac{U \cdot (1-U)}{1+U} \tag{1}$$

In this paper we are interested in scheduling multiple tasks under EDF, within each reserve. Still, it is trivial to show that the sufficient inflation, as a function of the sum U of the utilisations of the tasks served by the reserve, is given by the same expression (as long as the timeslot length S does not exceed the interarrival time of any task served by the reserve – a constraint also enforced herein). So as not to disrupt, the proof is in the Appendix (see Theorem 5).

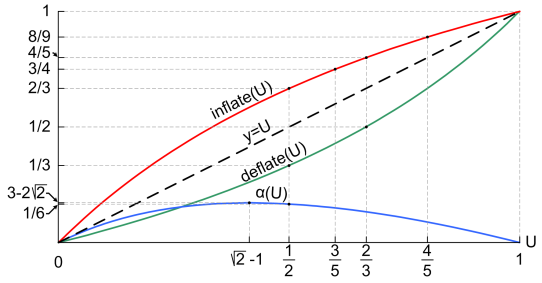

Figure 1. Plot of the functions α(U), inflate(U), deflate(U), with y=U as a reference.

For convenience later on, we introduce the function

$$\text{inflate}(U) = U + \alpha(U) = \frac{2 \cdot U}{1+U} \tag{2}$$

which expresses what size, as fraction of the timeslot length, a periodic reserve should be so as to be able to accommodate tasks of cumulative utilisation U. The inverse function

$$\text{deflate}(U) = \text{inflate}^{-1}(U) = \frac{U}{2-U} \tag{3}$$

expresses the maximum cumulative task utilisation that a periodic reserve of size U times the timeslot length may accommodate. For illustration purposes, see Figure 1.

2.3. On bin-packing

Bin-packing schemes have long been used for task assignment on partitioned multiprocessors. A popular such scheme, *First-Fit*, assigns tasks one by one to the lowest-indexed processor where each fits, subject to previous assignments (see [10], p. 124). We will define here a variant of First-Fit, for use with identical multiprocessors, with the property that it is impossible (assuming task utilisations do not exceed unity) for a task to fail to be assigned to some processor (subject to existing assignments) unless all processors are utilised by more than $\frac{1}{2}$.

Definition 1. *An ordered task set is in Heavy-First order (abbreviated HF) if and only if every task with utilisation higher than $\frac{1}{2}$ (i.e. a "heavy" task) precedes every task with utilisation $\frac{1}{2}$ or less (i.e. a "light" task).*

Multiple HF orderings might exist for a task set.

Definition 2. *The First-Fit – Heavy-First bin-packing algorithm (abbreviated FF-HF) is the First-Fit bin-packing algorithm with tasks considered in HF order.*

Our approach (described later) initially uses FF-HF bin-packing for task assignment until some task cannot be assigned (subject to existing assignments); it then attempts to accommodate remaining unassigned tasks via other mechanisms. So as to derive the utilisation bound of the algorithm we then need to know what fraction of system capacity is utilised, *before* the FF-HF bin-packing fails. Thus, in the context of task assignment on identical processors:

Definition 3. *The bin-packing bound of a bin-packing algorithm is the maximum valid lower bound on the utilised system capacity immediately prior to an unsuccessful attempt to assign a task (subject to assignments already made).*

Definition 4. *The per processor bin-packing bound of a bin-packing algorithm is the maximum valid lower bound for the minimum of the respective utilised capacities of all processors immediately prior to an unsuccessful attempt to assign a task (subject to assignments already made).*

Theorem 1. *The bin-packing bound and the per processor bin-packing bound of FF-HF are 50%.*

Proof: Assume task assignment over m identical processors according to the FF-HF algorithm. Task utilisations do not exceed unity. Then, if at some point during the procedure, some task τ_f with utilisation $u_f \stackrel{\text{def}}{=} \frac{C_f}{T_f}$ cannot be assigned to any processor, subject to assignments already made, two complementary possibilities exist:

- **Case 1:** $u_f > \frac{1}{2}$: That τ_f could not be assigned, implies that, immediately prior to the attempt to assign τ_f, every processor has at least one task assigned to it (or else some entirely unutilised processor would have existed to assign τ_f to – a contradiction). Moreover, since τ_f is heavy and tasks are considered in HF order, it follows that all previously assigned tasks were heavy as well. Thus, every processor is already utilised by more than 50% before attempting to assign τ_f. Therefore the entire system is also utilised by more than 50% *before* attempting to assign τ_f.
- **Case 2:** $u_f \leq \frac{1}{2}$: That τ_f could not be assigned implies that the unutilised capacity of every processor prior to the attempt is less than u_f. Equivalently, the utilised capacity of every processor, before attempting to assign τ_f, is more than $1-u_f$, in turn no less than $1-\frac{1}{2}=\frac{1}{2}$ (according to the assumption of the case). The entire system is thus also utilised by more than 50% *before* the attempt.

In either case, 50% is thus a lower bound both for individual processor utilisation and for the utilised system capacity before FF-HF can fail. That it is also, respectively, the greatest such lower bound may be shown via an example wherein, just prior to a failed attempt by FF-HF to assign a task (subject to existing assignments), all processors are utilised by $0.5+\epsilon>0.5$ (with $\epsilon\rightarrow 0^+$). Indeed, this occurs when given two processors and three tasks with utilisations $0.5+\epsilon$.

□

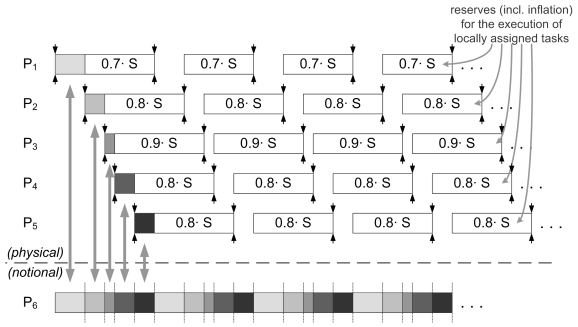

Figure 2. An example of a notional processor, implemented upon 5 physical processors.

2.4. On notional processors

A notional processor is a logical construct implemented upon multiple physical processors for the purposes of scheduling computational tasks. Essentially, it is a function providing a mapping, for every instant, to some (provably idle) physical processor, where a task *logically* treated as executing on the notional processor in question will actually be executing. For an intuitive example of the semantics see Figure 2. Formally, a notional processor is denoted by

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

with the following constraints:

- $a_0 = 0$ and $a_z \leq S$
- $\forall r \in \{1..z\}: \ a_{r-1} < a_r$
- $\forall r: \ h_r \in \{1..m\}$

and with the semantics that on time instant t, any task logically treated as executing on the notional processor in consideration is actually executing on processor P_{h_r}, where r is the integer for which the following holds:

$$a_r \leq t \text{ modulo } S < a_{r+1} \tag{4}$$

If $a_z = S$ then the notional processor is *full-capacity* (i.e. will always be mapped to some physical processor, on every instant). Else, it is fractional-capacity (i.e. periodically unavailable, akin to a periodic reserve).

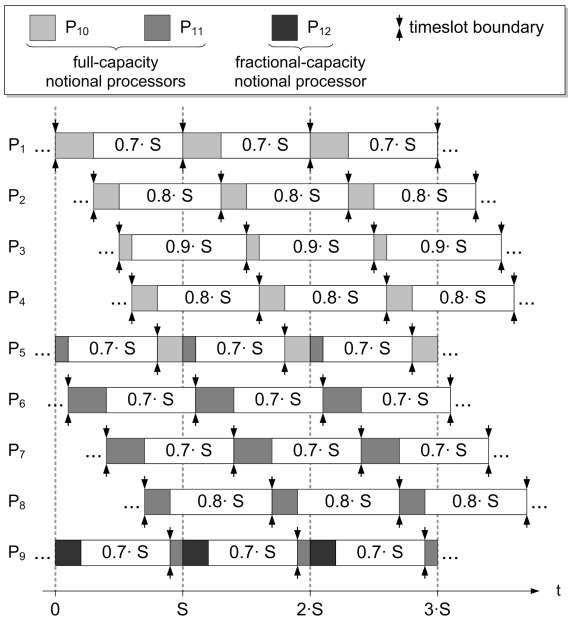

Figure 3. A system with multiple notional processors, including one of fractional capacity.

In terms of processing capacity, a full-capacity notional processor is "as good" as a physical one because it supplies processing power that is continuous in time (as it always maps to some physical processor, on any instant) and invariant (as the physical processors mapped-to are identical). In this respect it is unlike a typical server (which employs disjoint time windows to schedule its workload and may be preemptible), hence the new term ("notional processor"), for lack of an established term covering these semantics.

3. The new algorithm

The algorithm is structured in three stages:

- First, we assign tasks to physical processors, until we encounter a task which we cannot assign anywhere (subject to assignments already made).
- Then, we restrict the workload on each processor to execution within (appropriately sized) periodic reserves and organise the time intervals in between reserves on the physical processors into notional processors.
- Remaining tasks are assigned to notional processors.

We explore these stages in a more detailed manner:

In the first stage we assign tasks to physical processors using the FF-HF bin-packing algorithm (introduced in Section 2.3), until either all tasks have been assigned (in which case the algorithm completes) or we encounter a task which cannot be assigned to any processor (subject to assignments already made). In the latter case, because of the properties of the bin-packing scheme employed, each of the physical processors will be utilised by more than $\frac{1}{2}$ (by the tasks successfully assigned to each). (In the former case, no further action is necessary; we can simply use partitioned EDF.)

In the second stage, we select the value for an important, system-wide setting: the timeslot length S. It is set equal to the shortest of the interarrival times of all tasks (whether yet assigned or not). We then determine, for each physical processor P_p, what the length of the corresponding reserve x_p should be, as a fraction of the timeslot length, such that all tasks assigned to P_p will meet deadlines, if scheduled under EDF during this periodic reserve. This enforces, on every physical processor P_p, intervals of length x_p wherein tasks assigned to the processor may execute (under an EDF scheme) interleaved with "gaps" – intervals of length $S - x_p$. At this point we specify offsets for timeslot boundaries on adjacent processors such that, whenever the periodic "gap" on processor P_p ends, the "gap" on processor P_{p+1} begins. This provides a seamless supply of processing capacity that we structure into notional processors using the algorithm described in Figure 4. Figures 2 and 3 provide a more intuitive, visual explanation of how notional processors are implemented on top of physical processors.

In the third stage, we perform assignment of remaining tasks to notional processors, using First-Fit bin-packing (i.e. not necessarily FF-HF; any ordering will do). If all tasks can be assigned, the algorithm declares SUCCESS; otherwise, it declares FAILURE upon encountering the first task which cannot be assigned to any notional processor (subject to assignments already made).

4. On the utilisation bound of the algorithm

Because the algorithm actually performs two rounds of bin-packing (one over physical and one over notional processors), deriving its utilisation bound is not straightforward. In the general case, there exist m physical processors, m' full-capacity notional processors (indexed $m+1..m+m'$) and also a notional processor of fractional capacity F, with $0 \leq F < 1$. Then, the cumulative system utilisation is

$$U_{cml} = \sum_{p=1}^{m} U_p + \sum_{p=m+1}^{m+m'+1} U_p \qquad (5)$$

where the first and second term denote the sum of the utilisations of physical and notional processors (of either full or fractional capacity), respectively. Then, the system utilisation normalised by the number of physical processors is

```
1.  i,p,r,S,m_prime,cur_p,first_unassigned:integer;
2.  C[],T[]: array[1..n] of float;
3.  A[]: array[1..n] of integer;
4.  CAP[],U[],x[],O[]: array [1..m] of float;
5.  a[][]:array[m+1..m+m_prime+1][0..m] of float;
6.  h[][]:array[m+1..m+m_prime+1][0..m-1] of integer;
7.  unassigned_tasks_exist,stop: boolean;
8.  notional_CAP,end: float;
9.  read_task_parameters_from_input(); //C[], T[]
10. //stage 1 - - - - - - - - - - - - - - - - - - - - -
11. reindex_tasks_in_HF_order(); //C[], T[]
12. for p:= 1 to m do
13.    CAP[p]:=1; //full-capacity
14.    U[p]:=0;   //initially unutilised
15. end for
16. unassigned_tasks_exist:=TRUE;
17. for i:=1 to n do
18.    A[i]:=-1; //initially not assigned
19.    p:=1;
20.    while (p<=m) do
21.       if (U[p]+C[i]/T[i]<=CAP[p]) then
22.          A[i]:=p;
23.          U[p]:=U[p]+C[i]/T[i];
24.       else
25.          p:=p+1;
26.       end if
27.    end while
28.    if ((i==n) and (A[i]!=-1)) then
29.       unassigned_tasks_exist:=FALSE;
30.    end if
31.    if (A[i]==-1) then //was not assigned
32.       first_unassigned:=i;
33.       break;
34.    end if
35. end for
36. //stage 2 - - - - - - - - - - - - - - - - - - - -
37. S:=T[1];
38. for i:=1 to n do
39.    if (T[i]<S) then
40.       S:=T[i];
41.    end if
42. end for
43. O[1]:=notional_CAP:=0;
44. for p:=1 to m do
45.    x[p]:=inflate(U[p])*S;
46.    notional_CAP:=notional_CAP+(1-x[p]);
47. end for
48. for p:=1 to m-1 do
49.    O[p+1]=O[p]+(S-x[p]);
50. end for
51. m_prime:=floor(notional_CAP);
52. enlarge_arrays_U_and_A_and_CAP_by(m+m_prime+1);
53. for p:=m+1 to m+m_prime do
54.    CAP[p]:=1; //full-capacity notional CPU
55. end for
56. CAP[m+m_prime+1]:=deflate(notional_CAP-m_prime);
57. //now create notional CPUs
58. cur_p:=1;
59. for p := m+1 to m+m_prime+1 do
60.    stop:=FALSE;
61.    a[p][0]:=0;
62.    r:=1;
63.    while (stop==FALSE) do
64.       end:=S*inflate(CAP[p]);
65.       a[p][r]:=min(a[p][r-1]+S-x[cur_p],end);
66.       h[p][r-1]:=cur_p;
67.       r:=r+1;
68.       if (a[p][r]==end) then
69.          stop:=TRUE;
70.       else
71.          cur_p:=cur_p+1;
72.       end if
73.    end while
74. end for
75. //stage 3 - - - - - - - - - - - - - - - - - - - - -
76. for i:=first_unassigned to n do
77.    p:=m+1;
78.    while (p<=m+m_prime+1) do
79.       if (U[p]+C[i]/T[i]<=CAP[p]) then
80.          A[i]:=p;
81.          U[p]:=U[p]+C[i]/T[i];
82.       else
83.          p:=p+1;
84.       end if
85.    end while
86.    if ((i==n) && (A[i]!=-1)) then
87.       declare(FAILURE);
88.    end if
89. end for
90. declare(SUCCESS);
```

Figure 4. The offline task assignment and notional processor creation algorithm

$$U_{nrm} = \frac{U_{cml}}{m} \tag{6}$$

The cumulative capacity of the physical processors is always equal to m. That of the notional processors, however, depends on the utilisations of the physical processors:

$$\mathrm{CAP}^{notl}_{cml} = \sum_{p=1}^{m} (1 - \mathrm{inflate}(U_p)) \tag{7}$$

This capacity is structured, in the general case, as $m'{=}\lfloor \mathrm{CAP}^{notl}_{cml} \rfloor$ full-capacity notional processors and one notional processor of capacity $F = \mathrm{deflate}(\mathrm{CAP}^{notl}_{cml} - m')$.

Thus, how much additional utilisation can be accommodated by the notional processors depends on the outcome of the bin-packing over the physical processors and cannot be considered in isolation. Note also that, although notional processors do help (i.e. with accommodating additional tasks), their count tends to decrease the higher the utilisation of the physical processors, past the first round of bin-packing. Thus, it is not obvious whether having highly-utilised or less utilised physical processors (or some other scenario), past the first round of bin-packing, is the worst-case scenario (i.e. the one minimising the cumulative utilisation threshold past which the system might be unschedulable). So as to determine the utilisation bound of the algorithm however, this scenario has to be characterised. And for that, we need to determine (a lower bound on) how much utilisation, in the worst case, fits over a given number of notional processors (and a given capacity for the fractional notional processor, if one exists). This unconventional bin-packing problem (see Figure 6) can be formulated as:

Assume m' unit-capacity bins (i.e. processors) and one of capacity F, $0{\leq}F{<}1$. We perform First-Fit bin-packing with the fractional-capacity bin (i.e. processor) considered last. Assuming item sizes (i.e. task utilisations) in the range (0, 1], find the maximum threshold (as a function of F,

```
1.  //runs on every physical processor, on every
2.  //task arrival/completion or timeslot boundary
3.  procedure dispatcher is
4.  stay_idle: boolean;
5.  p, np: integer;
6.  t: float;
7.  p:= this_processor();
8.  t:=time_since_bootup();
9.  if (((t-O[p]+S) mod S) >= S-x[p]) then
10.    edf_schedule_from_queue(p);
11. else
12.    stay_idle:=TRUE;
13.    for np:=m+1 to m+m_prime+1 do
14.       for r:=0 to length(a[np])-2 do
15.          if ((a[np][r]<=t) and (t<a[np][r+1]))
16.             then break;
17.          end if
18.       end for
19.       if (h[np][r]==p) then
20.          stay_idle:=FALSE;
21.          break;
22.       end if
23.    end for
24.    if (stay_idle==TRUE)
25.       then do_nothing();
26.       else edf_schedule_from_queue(np);
27.    end if
28. end if
29. end procedure
```

Figure 5. The run-time dispatching algorithm

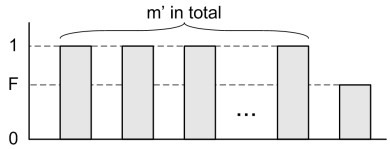

Figure 6. An unconventional bin-packing problem, due to the fractional-capacity bin.

m) such that any set of items (tasks) with cumulative size (utilisation) not above the threshold will always fit (i.e. manage to be assigned).

The problem is interesting in that the fractional-capacity bin might sometimes help, but not always. For example, its capacity might be smaller than even the smallest item.

So as to simplify the derivation, in those cases where $m'>0$, we (pessimistically) derive the corresponding utilisation bound for the case that the fractional-capacity processor is disregarded. (Intuitively, the utilisation bound cannot increase as a result. For proof, see Theorem 6 in the Appendix.) The unconventional bin-packing problem is then transformed to First-Fit bin-packing over m' identical processors, in which case the utilisation bound (as known from [12]) is $\frac{m'+1}{2}$. We proceed with a pessimistic lower-bound on the cumulative utilisation that we can "pack" into the system, subject to the outcome of the first round of bin-packing. For this, we rely on Theorem 2 via Lemma 1.

Lemma 1. *Assume two physical processors, P_a and P_b, respectively utilised by $U-x$, $U+x$, with $\frac{1}{2}\leq U-x<U+x\leq 1$. It then holds that $\alpha(U-x)+\alpha(U-x) < 2\cdot\alpha(U)$ (i.e. the cumulative processing capacity wasted as inflation so as to schedule the respective workloads on both processors within periodic reserves is less than what it would have been in the case that both P_a and P_b were utilised by U.)*

Proof: Follows from the fact that $\alpha(U)=\frac{U\cdot(1-U)}{1+U}$ is decreasing and strictly convex over $[\frac{1}{2}, 1]$. □

Theorem 2. *Assume m physical processors, P_1 to P_m, respectively utilised each by some U_p, $p\in\{1..m\}$ where $\frac{1}{2}\leq U_p\leq 1$. Then, the cumulative processing capacity sacrificed as inflation (so as to schedule the respective workloads on all physical processors within periodic reserves) cannot exceed what it would have been in the case that all processors were utilised by $U_\varphi \stackrel{\text{def}}{=} \frac{1}{m}\sum_{p=1}^{m} U_p$. Using notation: $\sum_{p=1}^{m}\alpha(U_p) \leq \sum_{p=1}^{m}\alpha(U_\varphi) = m\cdot\alpha(U_\varphi)$.*

Proof: Suppose that the claim was false. Then under some other scenario, the individual processor utilisations would not (all) be U_φ, but the average processor utilisation would still be U_φ and a greater fraction of the overall processing capacity would be lost on inflation, so as to schedule workloads within periodic reserves. We disprove this possibility:

If, among the m processors, we pick P_a, the processor with the lowest utilisation U_a and P_b, the processor with the highest utilisation U_b and modify the utilisations such that both P_a and P_b are utilised by $\frac{U_a+U_b}{2}$ (i.e. the average of the two original utilisations) then the following hold:

- The fraction of the utilised processing capacity of the m-processor array is the same, before and after.
- The fraction of the utilised processing capacity wasted as inflation can only increase (according to Lemma 1).

By repeating the above transformation, each time with the most and least utilised processor, we reach a state where all processors end up utilised arbitrarily close to U_φ (the average of the original processor utilisations) and the fraction of the overall processing capacity lost on inflation is higher. This contradicts the assumption that the claim is false. □

We now prove the utilisation bound for our algorithm (i.e. considering both rounds of bin-packing).

Theorem 3. *The utilisation bound of the above scheduling approach, is no less than $\frac{2}{3}$.*

Proof: Let U_φ denote, as before, the average of the utilisations of the m physical processors. From Theorem 2, the cumulative processing capacity wasted as inflation cannot exceed $m\cdot\alpha(U_\varphi)$, therefore the amount of processing capacity made into notional processors cannot be less than

$$m - m\cdot U_\varphi - m\cdot\alpha(U_\varphi) = m\cdot\big(1-\text{inflate}(U_\varphi)\big)$$

In turn, for m', the number of full-capacity notional processors, it holds that

$$m' \geq \lfloor m \cdot \big(1 - \text{inflate}(U_\varphi)\big) \rfloor \tag{8}$$

We explore three cases depending on the value of m:

- If $m \leq 3$, then if the sum of the utilisations of all tasks to be scheduled does not exceed $\frac{m+1}{2}$, it is known (see [12]) that First-Fit bin-packing (of which FF-HF is a special case) will assign all tasks to the m physical processors. Consequently, even if we would then opt to not use notional processors at all, the utilisation bound would be at least $\frac{3}{4}$ and $\frac{2}{3}$, respectively, for $m = 2$ and $m = 3$.
- If $m = 4$: If $U_\varphi > \frac{2}{3}$, then, irrespective of whether or not additional tasks are scheduled on the notional processors, the utilised system capacity cannot be less than $\frac{2}{3}$. Let us therefore focus on the remaining case where $U_\varphi \leq \frac{2}{3}$ and explore two subcases:
  - If $m' > 0$ (i.e. a full-capacity notional processor exists), then we can accommodate additional tasks of cumulative utilisation of at least unity. Thus, the utilisation bound is no less than $\frac{1}{m} \cdot (m \cdot U_\varphi + 1) > \frac{1}{4} \cdot (4 \cdot \frac{1}{2} + 1) = \frac{3}{4} > \frac{2}{3}$.
  - Else (i.e. if $m'=0$), we only have a notional processor of fractional capacity $F = \sum_{p=1}^{4}(1 - \text{inflate}(U_p))$. We can use that to schedule additional tasks of cumulative utilisation up to F – but only if $F > 1 - U_p$, $\forall p \in \{1, 2, 3, 4\}$. Otherwise, the fractional-capacity processor would not be able to accommodate whichever task the first round of bin-packing ended with (i.e. the one that could not be assigned). But, from Theorem 1, we know that $1 - U_p < 1 - \frac{1}{2} = \frac{1}{2}$, $\forall p \in \{1, 2, 3, 4\}$ whereas, from Theorem 2 and the fact that function inflate is strictly increasing and $U_\varphi \leq \frac{2}{3}$, we get $\sum_{p=1}^{4}(1 - \text{inflate}(U_p)) \geq \sum_{p=1}^{4}(1 - \text{inflate}(\frac{2}{3})) = \frac{4}{5}$. Therefore, the capacity F of the notional processor will be at least $\text{deflate}\left(\frac{4}{5}\right) = \frac{2}{3} > \frac{1}{2} > 1 - U_p$, $\forall p \in \{1, 2, 3, 4\}$, which means that we can, in any case, accommodate additional tasks of cumulative utilisation at least $\frac{2}{3}$. Therefore the utilisation bound is at least $\frac{1}{m} \cdot (m \cdot U_\varphi + F) > \frac{1}{4} \cdot (4 \cdot \frac{1}{2} + \frac{2}{3}) = \frac{2}{3}$.
- If $m \geq 5$: If $U_\varphi > \frac{2}{3}$, then, irrespective of whether or not additional tasks are scheduled on the notional processors, the utilised system capacity cannot be less than $\frac{2}{3}$. Let us therefore focus on the case where $U_\varphi \leq \frac{2}{3}$:
  Then, since the function inflate is strictly increasing and $\text{inflate}(\frac{2}{3}) = \frac{4}{5}$ and $U_\varphi \leq \frac{2}{3}$, and $m \geq 5$, it follows from Inequality 8 that $m' \geq 1$ (i.e. that there is at least one full-capacity notional processor).
  Then, we know (see [12]) that as long as the sum of the utilisations of remaining (i.e. from the first round of bin-packing) unassigned tasks does not exceed

$$\frac{1}{2} \cdot \Big(\lfloor m \cdot \big(1 - \text{inflate}(U_\varphi)\big) \rfloor + 1 \Big)$$

then all are eventually be assigned to some notional processor (thus the task set is schedulable by our algorithm). Therefore, a lower bound for the utilisation of the system, normalised by the number of physical processors, is

$$\begin{aligned} U &\geq \frac{1}{m} \cdot \Big(m \cdot U_\varphi + \frac{1}{2} \cdot \big(\lfloor m \cdot \big(1 - \text{inflate}(U_\varphi)\big) \rfloor + 1 \big) \Big) \\ &= U_\varphi + \frac{1}{2 \cdot m} \cdot \Big(\lfloor m \cdot \big(1 - \text{inflate}(U_\varphi)\big) \rfloor + 1 \Big) \end{aligned}$$

Since $\lfloor m \cdot \big(1 - \text{inflate}(U_\varphi)\big) \rfloor + 1 > m \cdot \big(1 - \text{inflate}(U_\varphi)\big)$, it holds that

$$\begin{aligned} U &> U_\varphi + \frac{1}{2 \cdot m} \cdot \Big(m \cdot \big(1 - \text{inflate}(U_\varphi)\big) \Big) \\ &= U_\varphi + \frac{1}{2} - \frac{1}{2} \cdot \text{inflate}(U_\varphi) = U_\varphi + \frac{1}{2} - \frac{U_\varphi}{1 + U_\varphi} \\ &= \frac{1}{2} + \frac{U_\varphi \cdot (1 + U_\varphi) - U_\varphi}{1 + U_\varphi} = \frac{1}{2} + \frac{U_\varphi^2}{1 + U_\varphi} \end{aligned} \tag{9}$$

The expression $\frac{U_\varphi^2}{1+U_\varphi}$ is an increasing function of U_φ over $(0,\ \infty)$. Additionally, we know from Theorem 1 that $U_\varphi > \frac{1}{2}$. Hence, over $(\frac{1}{2},\ 1]$, the minimum for the expression $\frac{U_\varphi^2}{1+U_\varphi}$ occurs at $U_\varphi = \frac{1}{2}$ and is equal to $\frac{1}{6}$. Combining this with Inequality 9 we obtain

$$U > \frac{1}{2} + \frac{1}{6} = \frac{2}{3} \tag{10}$$

Thus, irrespective of m, $\frac{2}{3}$ is a valid lower bound for the utilisation bound of our scheduling algorithm.

□

5. Bounds on preemption counts

Definition 5. *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*.

At run-time, there are five types of preemptions:

- type-α: Caused upon arrival of a task whose absolute deadline is earlier than that of some other task executing up to that point. The number of such preemptions, within a time interval Δt is bounded by the number of task arrivals within the same interval ($N_{arr}(\Delta t)$).

- type-β: These occur when a reserve ends, when the currently executing task, among those assigned to a physical processor, is preempted. The number of those preemptions, within a time interval Δt is at most $\lceil \frac{\Delta t}{S} \rceil$ per physical processor, thus at most $\lceil \frac{\Delta t}{S} \rceil \cdot m$ overall.
- type-γ: The migrations that occur when a reserve starts, when the notional processor currently "piggybacked" on top of the physical processor in consideration, migrates to another physical processor. Then, whichever task was logically on the notional processor at the time undergoes migration, in terms of the actual physical processor upon which it executes. In Figure 3, this occurs for notional processor P_{10} at t=$0.3 \cdot S$ (migration from P_1 to P_2), t=$0.5 \cdot S$ (migration from P_2 to P_3) and so on. The number of such preemptions within a time interval Δt is at most $\lceil \frac{\Delta t}{S} \rceil$ per physical processor, thus at most $\lceil \frac{\Delta t}{S} \rceil \cdot m$ overall.
- type-δ: The migrations that occur upon time instants which are integer multiples of the timeslot length S, when each full-capacity notional processor "rewinds" to using a lower-indexed physical processor. In Figure 3, this occurs on $t = S,\ 2 \cdot S, ...$, when notional processor P_{10} migrates from P_5 back to P_1. During a time interval Δt, at most $\lceil \frac{\Delta t}{S} \rceil \cdot m'$ of these may occur.
- type-ϵ: The preemptions that occur each time that the single fractional-capacity notional processor (if one exists) becomes unavailable. In the example of Figure 3, this occurs at $t = 0.2 \cdot S,\ 1.2 \cdot S, 2.2 \cdot S...$ During a time interval Δt (and only if a fractional-capacity notional processor exists), there are at most $\lceil \frac{\Delta t}{S} \rceil$ such preemptions.

Thus, during an interval of length Δt, overall preemptions can be at most

$$N_{preempt}(\Delta t) = \underbrace{N_{arr}(\Delta t)}_{\text{type-}\alpha} + \underbrace{\left\lceil \frac{\Delta t}{S} \right\rceil \cdot m}_{\text{type-}\beta} + \underbrace{\left\lceil \frac{\Delta t}{S} \right\rceil \cdot m}_{\text{type-}\gamma} + \underbrace{\left\lceil \frac{\Delta t}{S} \right\rceil \cdot m'}_{\text{type-}\delta} + \underbrace{\left\lceil \frac{\Delta t}{S} \right\rceil \cdot m'_{fr}}_{\text{type-}\epsilon}$$
$$= N_{arr}(\Delta t) + \left\lceil \frac{\Delta t}{S} \right\rceil \cdot (2 \cdot m + m' + m'_{fr}) \quad (11)$$

where m'_{fr} is the number of fractional-capacity notional processors (which can only be either 0 or 1) and m' is the number of full-capacity notional processors. We proceed to eliminate m', m'_{fr} from the above expression.

Theorem 4. *The number of notional processors (full-capacity or fractional capacity, irrespective) cannot exceed $\lceil \frac{m}{3} \rceil$, when the first-round of bin-packing is performed according to the FF-HF algorithm.*

Proof: If the cumulative capacity available, past the first-round of bin-packing, for structuring into notional processors is $\mathrm{CAP}^{notl}_{cml}$, out of that capacity, we can craft:

- $m' = \lfloor \mathrm{CAP}^{notl}_{cml} \rfloor$ full-capacity notional processors
- $m'_{fr} = \lceil \mathrm{CAP}^{notl}_{cml} - m' \rceil$ fractional capacity processor(s) of capacity $\mathrm{deflate}(\mathrm{CAP}^{notl}_{cml} - m')$

Thus:

$$m' + m'_{fr} = \lceil \mathrm{CAP}^{notl}_{cml} \rceil \quad (12)$$

Therefore, $m' + m'_{fr}$ is maximised when $\mathrm{CAP}^{notl}_{cml}$ is maximised. As earlier stated (remember Equation 7),

$$\mathrm{CAP}^{notl}_{cml} = \sum_{p=1}^{m} (1 - \mathrm{inflate}(U_p)) = m - \sum_{p=1}^{m} \mathrm{inflate}(U_p) \quad (13)$$

However, we know from Theorem 1 that $U_p > \frac{1}{2},\ \forall p \in [1..m]$. We also know that the function inflate is increasing. Using both of these facts and Equation 13, we conclude that

$$\mathrm{CAP}^{notl}_{cml} < m - \sum_{p=1}^{m} \mathrm{inflate}\left(\frac{1}{2}\right) = m - m \cdot \frac{2}{3} = \frac{m}{3} \quad (14)$$

Equation 12 and Inequality 14 combined yield:

$$m' + m'_{fr} \leq \left\lceil \frac{m}{3} \right\rceil \quad (15)$$

□

Directly applying Theorem 4 to Inequality 11 yields

$$N_{preempt}(\Delta t) \leq N_{arr}(\Delta t) + \left\lceil \frac{\Delta t}{S} \right\rceil \cdot \left(2 \cdot m + \left\lceil \frac{m}{3} \right\rceil\right) \quad (16)$$

as an upper bound for the number of preemptions during any interval of length Δt.

6. Discussion and conclusions

All task sets with utilisations up to $\frac{1}{2}$ are schedulable under partitioned EDF (using First-Fit bin-packing) with at most $N_{arr}(\Delta t)$ preemptions over an interval of length Δt.

Any task is schedulable by the algorithm in [3] if integer parameter δ is set to a sufficiently high value. The utilisation bound for that algorithm as a function of δ is given by:

$$\mathrm{UB}_{\text{ECRTS 2008}}(\delta) = 4 \cdot \left(\sqrt{\delta \cdot (\delta + 1)} - \delta\right) - 1$$

One should choose the smallest acceptable δ, to avoid needless preemptions, as the upper bound on preemptions, within an interval of length Δt for the algorithm in [3] is an increasing function of δ:

$$N^{\text{ECRTS 2008}}_{preempt}(\Delta t) = N_{arr}(\Delta t) + 3 \cdot m \cdot \delta \cdot \left\lceil \frac{\Delta t}{S} \right\rceil + 2$$

By comparison, for the algorithm formulated herein, the upper bound for preemptions is given by

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

We thus note that (since $2 \cdot m + \lceil \frac{m}{3} \rceil \leq 3 \cdot m \cdot \delta$, $\forall m \in \{1,\ 2,\ ...\}$, $\forall \delta \in \{1,\ 2,\ ...\}$) the algorithm formulated herein is able to schedule task sets with utilisations up to $66.\bar{6}\%$ with fewer preemptions than even the most "preemption-light" setting (δ=1) of the algorithm from [3], which has a utilisation bound of 65.7% only.

Note also that to potentially further reduce preemptions we can retroactively (i.e. past assignment of all tasks), with no loss of schedulability, increase the timeslot length to

$$S = \min_{\substack{i:\ \tau_i \text{ assigned to physical or} \\ \text{fractional notional processor}}} \{T_i\} \geq \min_{\forall i} \{T_i\}$$

whereas in [3] by comparison, $S = \min_{\forall i} \{T_i\}$, for $\delta = 1$. Thus, one may sometimes use longer timeslots than the algorithm in [3] (thus further reducing preemptions) with no detriment to schedulability. To leverage this, we propose for the second bin-packing round, assignment of remaining tasks in order of increasing interarrival time. If $m' > 0$, this heuristic prevents assignment of tasks with short interarrival times to the fractional-capacity notional processor.

Finally, we note that task sets of higher utilisation than the utilisation bound of $66.\bar{6}\%$ might still be schedulable under the algorithm introduced. Therefore, we advocate the following approach, when faced with the problem of scheduling task sets with utilisations greater than $\frac{1}{2}$:

- First try scheduling under partitioned EDF;
- if that fails, try with the algorithm formulated herein;
- if that fails, use [3], with an appropriate value for δ.

Consideration of scheduling algorithms in this order ensures schedulability with as few preemptions as necessary.

To conclude, our algorithm, with a utilisation bound of $66.\bar{6}\%$, is the most "preemption-light" (in terms of proven upper bounds on preemptions) of all known multiprocessor scheduling algorithms with utilisation bounds over 50%.

Acknowledgements

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

References

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

[2] 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.

[3] B. Andersson and K. Bletsas. Sporadic Multiprocessor Scheduling with Few Preemptions. In *20th Euromicro Conference on Real-Time Systems*, pages 243–252, 2008.

[4] B. Andersson and E. Tovar. Multiprocessor scheduling with few preemptions. In *Proceedings of the 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 322–334, 2006.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co, 1979.

[11] 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)*, p. 441–450, 2007.

[12] 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*, p. 25–33, 2000.

[13] 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. *Sufficient inflation for a periodic reserve to accommodate implicit-deadline tasks of cumulative utilisation U, under EDF, is given by* $\frac{U \cdot (1-U)}{1+U}$*, if the timeslot length* S *does not exceed 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, throughout all sub-intervals of $[t_m - L,\ t_m)$ which lie within the periodic reserve, the processor 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:

$$t_d > t^{\varphi} \tag{17}$$

Regarding t_d, it follows from [6] that

$$t_d \leq \sum_{\tau_i \in \tau} \left\lfloor \frac{L}{T_i} \right\rfloor \cdot C_i \overset{(17)}{\Longrightarrow} \sum_{\tau_i \in \tau} \left\lfloor \frac{L}{T_i} \right\rfloor \cdot C_i > t^{\varphi} \tag{18}$$

At this point we note that

$$\sum_{\tau_i \in \tau} \left\lfloor \frac{L}{T_i} \right\rfloor \cdot C_i \leq \sum_{\tau_i \in \tau} \left(\frac{L}{T_i} \cdot C_i \right) = L \cdot \sum_{\tau_i \in \tau} \frac{C_i}{T_i} = L \cdot U \tag{19}$$

which leads us to observe that

$$(18) \overset{(19)}{\Longrightarrow} L \cdot U > t^{\varphi} \Rightarrow U > \frac{t^{\varphi}}{L} \tag{20}$$

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 $(U{+}\alpha(U)){\cdot}S$ (corresponding to the reserves), interleaved by time windows of length $S - (U + \alpha(U)) \cdot S$, when the processor is unavailable to tasks served by the reserve. 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 $L = S + \big(S - (U{+}\alpha(U)){\cdot}S\big)$ (because it ends just as the next reserve begins). In that case, $t^{\varphi} = (U{+}\alpha(U)){\cdot}S$ and

$$\frac{t^{\varphi}}{L} = \frac{(U + \alpha(U)) \cdot S}{S + \big(S - (U + \alpha(U)) \cdot S\big)} = \frac{U + \alpha(U)}{2 - U - \alpha(U)} \tag{22}$$

Inequality 21 and Equation 22 combined yield:

$$\begin{aligned} U &\leq \frac{U + \alpha(U)}{2 - U - \alpha(U)} \\ \Leftrightarrow 2 \cdot U - U^2 - U \cdot \alpha(U) &\leq U + \alpha(U) \\ \Leftrightarrow \alpha(U) &\geq \frac{U \cdot (1 - U)}{1 + U} \end{aligned} \tag{23}$$

which proves the theorem. □

Theorem 6. *Consider a computing platform* Π *consisting of* μ *identical processors indexed* $1..\mu$ *and another computing platform* Π' *consisting of* μ *processors (identical to those of* Π *and indexed* $1..\mu$*) plus one additional processor (indexed* $\mu + 1$*) of fractional capacity* f*, with* $0 \leq f < 1$*. Let* τ' *denote a set of tasks with individual utilisations no more than unity. Then, for any given HF ordering of tasks in* τ'*, if the First-Fit bin-packing algorithm fails to assign every task to some processor over* Π'*, it will also fail when attempting to do so over* Π*, if using the same task ordering.*

Proof: Let us simulate independently First-Fit bin-packing (i) over Π' and (ii) over Π. In both cases, we use the same task set τ' and the same ordering (effectively, "cloning" τ'). However, the simulations are to take place in the following "lock-step" manner:

Within every step, after every simulated assignment (or assignment attempt) over Π', we simulate the assignment (or assignment attempt) of the same task over Π. Then, as a next step, we proceed with the next task over Π' and so on.

Due to the First-Fit scheme, a processor is only ever considered as an assignment target, if all lower-indexed processors have been considered and ruled out as assignment targets (subject to assignments already made) for the task in consideration. Thus, up to the point where only full-capacity processors have been considered as assignment targets in Π', the assignments over Π mimic those in Π'.

If then, upon trying to assign a task over Π', it cannot be assigned to any of the full-capacity processors, subject to assignments already made, then the following hold:

- The fractional-capacity processor in Π' will be next considered (and the assignment might succeed or fail).
- During the corresponding assignment attempt over Π, the attempted assignment of the task in consideration to each of the μ processors in Π (all full-capacity) will fail and the algorithm will hence declare failure.

Note also that the algorithm cannot declare failure, during assignment over Π' unless, at some point, the highest-indexed processor (i.e. that of fractional capacity) is considered as an assignment target and the assignment fails. □