Pfairness for late released tasks, with constrained deadlines

Sadouanouan Malo*[1,2], Annie Choquet-Geniet[1] and Moustapha Bikienga[2]

[1]University of Poitiers. Laboratory of Applied Computer Science
1 Av. Clément Ader BP 40109-86961 Futuroscope Chasseneuil-France
[2]Polytechnic University of Bobo Dioulasso.Information Technology High School.
01 BP 1091 Bobo Dioulasso 01
E-mail: sadouanouan.malo@ensma.fr, annie.geniet@univ-poitiers.fr, bmoustaph@yahoo.fr
* Corresponding author

Abstract: Pfair scheduling has usually been applied in the context of synchronous periodic task systems with implicit deadlines. This paper addresses the problem of scheduling asynchronous hard real-time tasks with constrained deadlines using a Pfair strategy on multiprocessor systems. First, we extend the notion of Pfairness to the context of asynchronous tasks with constrained deadline. Then we investigate feasibility conditions, we propose a rather efficient one and we illustrate the relevance of our criteria through some simulations.

Keywords: Scheduling - Fairness - Pfair feasibility

1 Introduction

Multiprocessor systems, characterized by their performance and reliability, have evolved to be powerful computing platforms widely used in many real-time applications. In this paper we consider global scheduling of hard real-time tasks and we assume that preemption and interprocessor migration are permitted and parallelism forbidden (at any time, a task can run on at most one processor).
In this context, it has been shown that no on-line scheduling algorithm can be optimal Mok and Dertouzos (1978); Dertouzos and Mok (1989). Baruah et al. (1995, 1996) proved that the problem of optimally scheduling synchronous periodic tasks with implicit deadlines on identical multiprocessor platforms could be solved at run-time in polynomial time using Pfair scheduling algorithms. Under Pfair, each periodic task is required to progress at steady rate. Each task is assigned a weight corresponding to its utilisation factor that represents the part of the activity of processors dedicated to this task. Because of their efficiency (Pfair

strategies are optimal) and because they are of interest for e.g. multimedia applications, Pfair scheduling algorithms have been widely investigated last years Anderson et al. (2000, 2004); A.Srinivasan et al. (2003). In all these works, the notion of Pfairness is defined in the context of synchronous tasks with implicit deadlines. In Anderson et al. (2004); Devi and Anderson (2005), asynchronous systems are considered, where asynchronism means that some of the first task slots do not take place. But the first job of any task is still assumed to be released at time 0. In Anderson et al. (2000), sporadic tasks are considered: periods correspond only to the minimum elapsed time between two consecutive releases. They introduce the intra-sporadic model, where the release time of each task slot can be chosen. We consider here a slightly different notion of asynchronism: the first release times of the different tasks, i.e. the release time of the first task slot, are no more assumed to be equal. But all the task slots are assumed to occur, and their release times cannot be chosen arbitrarily . To shift a task may e.g. be useful in order to take some precedence relations into account. Furthermore, requiring tasks to have periods equal to deadlines restricts the application of Pfairness in practice. The problem of scheduling tasks with arbitrary deadlines in a Pfair way on multiprocessors has been addressed in Ramamurthy (2002) and a static-priority scheme to schedule a set of such tasks has been presented. In this paper we investigate extension of Pfairness in order to design Pfair scheduling schemes for asynchronous tasks with deadlines less than or equal to periods. As far as we know, others approaches don't jointly consider asynchronous tasks and constrained deadlines. We first extend and adapt Pfairness definition to this new context and then we propose a feasibility condition. Since there exists no optimal strategy in this general case, we cannot propose a necessary and sufficient condition. We propose here only a sufficient but rather effective condition and then we present some simulation results.
The remainder of the paper is organized as follows. In section 2 we formally define the Pfair multiprocessor scheduling problem. In section 3 we prove that Pfair algorithm exists for any periodic task set and we give sufficient feasibility conditions. In section 4 we present the simulations results. Conclusions and perspectives are given section 5.

2 Pfair scheduling

In this section we present Pfair scheduling for synchronous tasks with implicit deadlines. We adopt the following notations: for any real number x, $\lfloor x \rfloor$ is the integer immediately below or equal to x and $\lceil x \rceil$ is the integer immediately above or equal to x.
In the further, slot t denotes the time interval [t, t+1). We assume that processors are allocated for integral number of slots, thus a task cannot be preempted within a slot.

2.1 Basic definitions

We use the classical task model.

Definition 1. *A task $\tau_i =< r_i, C_i, D_i, T_i >$ is characterized by four parameters: its*

first release time r_i, its per period worst case execution time C_i, its relative deadline D_i and its period T_i.
The weight of task τ_i is $U_i = \frac{C_i}{T_i}$ and its density is $CH_i = \frac{C_i}{D_i}$.

We have $r_i \geq 0$ and we assume that C_i, D_i and T_i are integral and verify $C_i \leq D_i \leq T_i$. Thus we have $0 < U_i \leq 1$. In this section we consider a set Γ of tasks such that $r_i = 0$ and $D_i = T_i$.

Definition 2. *A m-processor schedule for a task set Γ is a function $S : \Gamma \times \mathbb{N} \to \{0, 1\}$, such that $\forall t \in \mathbb{N} : \sum_{\tau_i \in \Gamma} S(\tau_i, t) \leq m$ with $S(\tau_i, t) = 1$ if task τ_i is scheduled in slot $[t, t+1)$ and $S(\tau_i, t) = 0$ else.*

For the remainder of the paper, we simply use schedule to denote an m-processor schedule of Γ. A fair schedule is approximately an ideal fluid schedule such that, at any time t, each task has been processed for $\omega_i(t) = U_i \times t$ processor time units. Now, since processor time is allocated in integral number of slots, the ideal behaviour is approximated by either the integer directly above or directly beyond. The deviation from this fluid schedule is formally captured by the concept of *lag*. The **lag** of a task τ_i at time t in a schedule S is given by $lag(S, \tau_i, t) = \omega_i(t) - \sum_{j=0}^{j=t-1} S(\tau_i, t)$. A schedule S is said to be **Pfair** iff

$$\forall \tau_i, t : \tau_i \in \Gamma, t \in \mathbb{N}, -1 < lag(\tau_i, t) < 1$$

Informally, the allocation error associated with each task must always be less than one slot. This condition can be graphically interpreted in the following way: from ideal CPU service of the task defined by $ideal(t) = U_i * t$, we draw two limit lines defined by $W^+(t) = U_i * t + 1$ and $W_- = U_i * t - 1$. Then, the broken line representing the actual CPU service of the task must remain strictly between both limit lines (see figure 1).

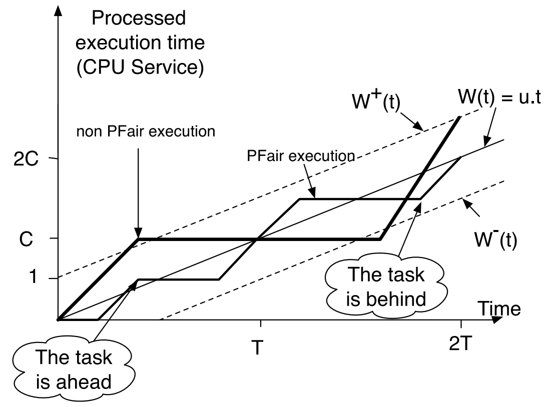

Figure 1 Pfair and non Pfair behaviours

2.2 *Subtasks*

For implementation reasons, each task τ_i is split into series of quantum-length *subtasks* (i.e. with execution time equal to one slot). The j^{th} *subtask* ($j \geq 0$) of task τ_i is denoted τ_i^j. Each subtask τ_i^j has a pseudo-release time r_i^j and a pseudo-deadline d_i^j, deduced from the lag inequations: $r_i^j = \left\lfloor \frac{j}{U_i} \right\rfloor$ and $d_i^j = \left\lceil \frac{j+1}{U_i} \right\rceil$. τ_i^j must be scheduled in the interval $I_i^j = [r_l^j, d_i^j)$, called its **feasibility window**. Note that r_i^{j+1} is either $d_i^j - 1$ or d_i^j. Thus, two consecutive windows of the same task either overlap by one slot or are disjoint . Consider as an example the task $\tau_i =< 0, 3, 5, 5 >$. It is split each period into 3 subtasks. For the first period, τ_i^0 must be scheduled within the time interval $[0, 2)$,τ_i^1within $[1, 4)$ and τ_i^2 within $[3, 5)$. Feasibility windows of the six first subtasks (two first periods of the task) are depicted in figure 2. We present a possible Pfair execution. We notice that the curve of the actual CPU service of the task is effectively located between the limit lines. In the first period of activity, the task is scheduled at times 0, 2 and 3.

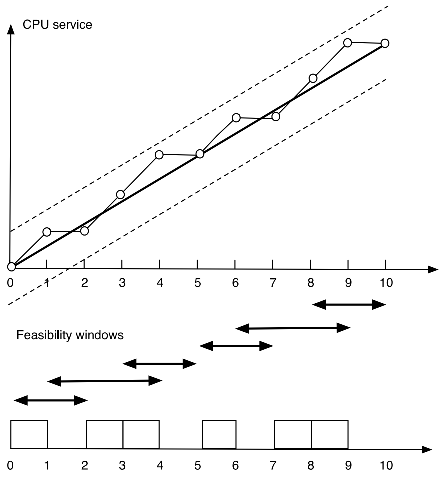

Figure 2 A Pfair schedule for task $< 0, 3, 5, 5 >$ - Feasibility windows

2.3 Pfair strategy

Pfair algorithm has the following high-level structure: at each time $t \geq 0$, a dynamic priority is assigned to each task and the m highest-priority tasks are scheduled in slot t. The priorities rely on fairness. With respect to a Pfair schedule S at time t, we say that task τ_i is **ahead** if the task has been processed a little bit more than in the ideal case $(lag(S, \tau_i, t) < 0)$, **behind** if it has been processed a little bit less than in the ideal case $(lag(S, \tau_i, t) > 0)$ and **punctual** if it has been processed exactly as in the ideal case (see figure 1). At any time t, Pfair strategies split tasks into three categories:

- A **Urgent** task is a task which is behind and which would be late if it were not processed at this time. Such a task must be processed at time t, else the Pfairness condition would be violated,
- a **Tnegru** task is ahead and would be in advance if it were processed at this time. Such a task must not be processed at time t, else the Pfairness condition would be violated,

- a **Contending** task is neither Urgent, nor Tnegru. The Pfairness will neither be violated if it is processed nor if it is not.

Pfair algorithms can be summarized as follows:

1. All urgent tasks are scheduled.
2. Contending tasks are sorted.
3. The remaining processors are allocated to the highest-priority contending tasks.

Three Pfair scheduling algorithms are known to be optimal on an arbitrary number of processors: PF, PD and PD^2 Baruah et al. (1996, 1995); Anderson et al. (2000, 2004). These algorithms differ in the choice of tie-breaking rules. One tie-break parameter that is common to all three algorithms is the *successor bit*, which is defined as follows: $b(\tau_i^j) = b_i^j = d_i^j - r_i^{j+1} = \left\lceil \frac{l}{U_i} \right\rceil - \left\lfloor \frac{l}{U_i} \right\rfloor$. The successor bit of a sub-task is equal to 1 if it feasibility window overlaps the feasibility window of the next sub-task, and is equal to 0 if both windows are disjoint.
We present here the PF algorithm. Tasks are split into unitary subtasks which are prioritized as follows: at time t, if subtasks τ_i^u and τ_j^v are both ready, then τ_i^u has higher priority than τ_j^v , denoted $\tau_i^u \succ \tau_j^v$ if one of the following condition holds:
(i) $d_i^u < d_j^v$.
(ii) $d_i^u = d_j^v$ and $b_i^u > b_j^v$.
(iii) $d_i^u = d_j^v$, $b_i^u = b_j^v = 1$, and $\tau_i^{u+1} \succ \tau_j^{v+1}$.
If neither subtask has priority over the other, then tie can be broken arbitrarily. At the beginning of each slot, the m highest priority subtasks are selected to run in that slot. In the remainder of the paper, we will consider the algorithm PF to illustrate our results.

2.4 Feasibility

As mentioned, Pfair scheduling is very efficient. Furthermore, there exists a very simple characterization of feasible task sets. This is stated by the following theorem Baruah et al. (1996); Anderson et al. (2004).

Theorem 1. *The algorithms PF, PD and PD^2 are optimal for systems of synchronous independent tasks with implicit deadlines. Moreover, a synchronous periodic task set with implicit deadlines Γ has a Pfair schedule on m processors if and only if $\sum_{\tau_i \in \Gamma} \frac{C_i}{T_i} \leq m$.*

3 Extension of Pfairness

Our aim is now to get rid of the restrictive assumptions on task sets. In this paper we want to prove that fairness can be extended to all independent periodic task sets. We have first carried out some simulations. For that purpose we have first developed a Pfair scheduling simulator composed of two parts: a real-time task generator and a multiprocessor scheduler. The number of processors is not limited and generated task sets are such that $\sum_{i=1}^{i=n} \frac{C_i}{T_i} \leq m$. The scheduler implements the algorithm PF. Results of our simulations are summarized in section 4. These simulations lead to two establishments: firstly, if deadlines less than or equal to periods are considered, if $\sum_{i=1}^{i=n} \frac{C_i}{D_i} \leq m$, the task set is feasible on m processors and secondly if asynchronous tasks with implicit deadlines are considered, Baruah's condition still holds. This leads us to infer the following result:

Theorem 2. *Given a periodic task set* Γ, *if* $\sum_{i=1}^{i=n} \frac{C_i}{D_i} \leq m$ *then* Γ *has a valid Pfair schedule on m processors over any time interval* $[0, t)$.

Before we prove the theorem, we first extend the notion of Pfairness. We consider *periodic task sets* such that, if Γ is a task set, then $\forall \tau_i \in \Gamma$, $\tau_i =< r_i, C_i, D_i, T_i >$ with $r_i \geq 0$ and $D_i \leq T_i$. T denotes the **hyperperiod** of the task set, defined as $T = LCM(T_i)_{\tau_i \in \Gamma}$. Pfairness is extended in the following way: in an ideal schedule of a periodic task set, each task τ_i must have received at time t, $\omega_i(t)$ processor time units (see figure 3). $\omega_i(t)$ is such that:

$$\omega_i(t) = \begin{cases} 0 \; if \; t \in [0, r_i) \\ k * C_i + \frac{C_i}{D_i} * (t - k * T_i - r_i) \; if \\ t \in [k * T_i + r_i, k * T_i + D_i + r_i) \\ (k+1) * C_i \\ if \; t \in [k * T_i + D_i + r_i, (k+1) * T_i + r_i). \end{cases}$$

where $k = \left\lfloor \frac{t}{T_i} \right\rfloor$ represents the instance number of the pending instance of the task. A schedule S is then Pfair if and only if $\forall \tau_i \in \Gamma, \forall t \in \mathbb{N} : -1 < lag(S, \tau_i, t) < 1$ where $lag(S, \tau_i, t) = \omega_i(t) - \sum_{j=0}^{j=t-1} S(\tau_i, j)$. We also redefine the feasibility windows of a task. There is no window within idle periods, and we define the j^{th} window as $I_i^j = [r_i^j, {}_i^j)$ with $r_i^j = r_i + k * T_i + \left\lfloor \frac{j - k.C_i}{CH_i} \right\rfloor$ and $d_i^j = r_i + k.T_i + \left\lceil \frac{j+1-k.C_i}{CH_i} \right\rceil$

The proof of the theorem we give is an adaptation of the proof presented in Baruah et al. (1996) to prove theorem 1 for PF. It is based on graph theory. We prove that a Pfair schedule exists on any time interval $[0, L)$. In the further, $CTR_{\tau_i}(t)$ denotes the complete processor demand of all sub-tasks of task τ_i whose feasibility intervals are included in $[0, t]$. Thus, $CTR_{\tau_i}(t)$ is equal to j where j is such that $d_i^j \leq t < d_i^{j+1}$. We first define a weighted digraph G and prove that if G has an integral flow of size $\sum_{\tau_i \in \Gamma} CTR_{\tau_i}(L)$ then the task set Γ has a Pfair schedule.

Definition 3. *The Pfair-graph is the weighted digraph* $G(L)$ *is defined as* $G(L) = (V, E)$ *with:*
$V = V_0 \cup V_1 \cup V_2 \cup V_3 \cup V_4 \cup V_5$ *and* $E = E_0 \cup E_1 \cup E_2 \cup E_3 \cup E_4$
$V_0 = \{< source >\}$
$V_1 = \{< 1, \tau_i >, \tau_i \in \Gamma\}$

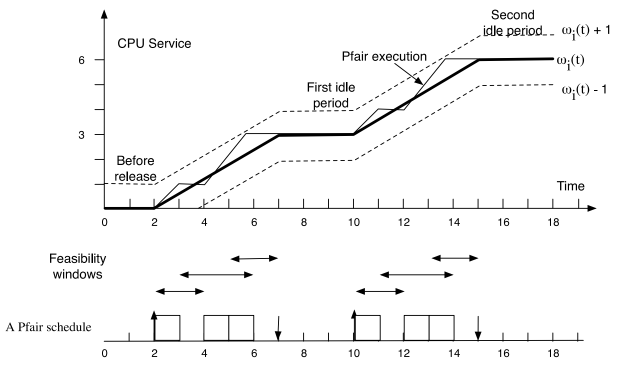

Figure 3 Ideal progression of allocated CPU time for a late released task with constrained deadline - Idle periods

$V_2 = \{< 2, \tau_{i0}, 0 >, i = 1..n \ s.t. \ r_i > 0\} \cup$
$\{< 2, \tau_i^j >, (i,j) \ s.t. \ i = 1..n, j \in [0, CTR_{\tau_i}(L))\} \cup \{< 2, \tau_{i0}, j >, (i,j) \ s.t. \ i = 1..n, j \in [1, \left\lfloor \frac{L-r_i}{T_i} \right\rfloor]\}$
$V_3 = \{< 3, \tau_{i0}, t >, (i,t) \ s.t. \ i = 1..n, t \in [0, r_i)\} \cup$
$\{< 3, \tau_i, t >, (i,t) \ s.t. \ i = 1..n, t \in [k.T_i + r_i, k.T_i + r_i + D_i), \ with \ 1 \leq k \leq \left\lfloor \frac{L-r_i}{T_i} \right\rfloor\} \cup$
$\{< 3, \tau_{i0}, t >, (i,t) \ s.t. \ i = 1..n, t \in [k.T_i + D_i, (k+1).T_i) \ with \ 1 \leq k \leq \left\lfloor \frac{L-r_i}{T_i} \right\rfloor\}$
$V_4 = \{< 4, t >, t \in [0, L)\}$
$V_5 = \{< sink >\}$.
Edges and capacities are defined by:
$E_0 = \{(< source >, < 1, \tau_i >, CTR_{\tau_i}(L)), i = 1..n\}$
$E_1 = \{(< 1, \tau_i >, < 2, \tau_{i0}, 0 >, 0), i = 1..n \ s.t. \ r_i > 0\} \cup$
$\{(< 1, \tau_i >, < 2, \tau_i^j >, 1), j \in [0, CTR_{\tau_i}(L))\} \cup$
$\{(< 1, \tau_i >, < 2, \tau_{i0}, j >, 0), (i,j) \ s.t. \ i = 1..n, j \in \left[1, \left\lfloor \frac{L-r_i}{T_i} \right\rfloor\right]\}$
$E_2 = \{(< 2, \tau_{i0}, 0 >, < 3, \tau_{i0}, t >, 0), (i,t) \ s.t. \ i = 1..n, t \in [0, r_i)\} \cup$
$\{(< 2, \tau_i^j >, < 3, \tau_i, t >, 1), (i,j,t) \ s.t. \ i = 1..nj \in [0, CTR_{\tau_i}(L)), t \in [r_i^j, d_i^j)\} \cup$
$\{(< 2, \tau_{i0}, j >, < 3, \tau_i, t >, 0)(i,j,t) \ s.t. \ i = 1..n, j \in [1, \left\lceil \frac{L-r_i}{T_i} \right\rceil, t \in [k.T_i + r_i + D_i, (k+1).T_i + r_i) \ with \ 1 \leq k \leq \left\lfloor \frac{L-r_i}{T_i} \right\rfloor]t\}$
$E_3 = \{(< 3, \tau_{i0}, t >, < 4, t >, 0), t \in [0, r_i)\} \cup$
$\{(< 3, \tau_i, t >, < 4, t >, 1), t \in [k.T_i + r_i, k.T_i + r_i + D_i) \ with \ 1 \leq k \leq \left\lfloor \frac{L-r_i}{T_i} \right\rfloor\} \cup$
$\{(< 3, \tau_{i0}, t >, < 4, t >, 0), (i,t) \ s.t. \ i = 1..n, t \in [k.T_i + r_i + D_i, (k+1).T_i) \ with \ 1 \leq k \leq \left\lfloor \frac{L-r_i}{T_i} \right\rfloor\}$.
$E_4 = \{(< 4, t >, < sink >, m), t \in [0, L)\}$.

As an example, consider the synchronous task set Γ with constrained deadlines composed of two tasks: $\Gamma = \{< 0, 3, 5, 8 >, < 0, 2, 3, 4 >\}$. We chose a synchronous task set in order to easily represent the Pfair-graph. Indeed, because of the cyclicity property of synchronous task systems, we just have to study the time interval $[0, 8]$. Figure 4 gives the resulting Pfair-graph. The intuition behind the structure of the graph G is the following.
V_1 contains one node for each task τ_i and the capacity of the edge from the source

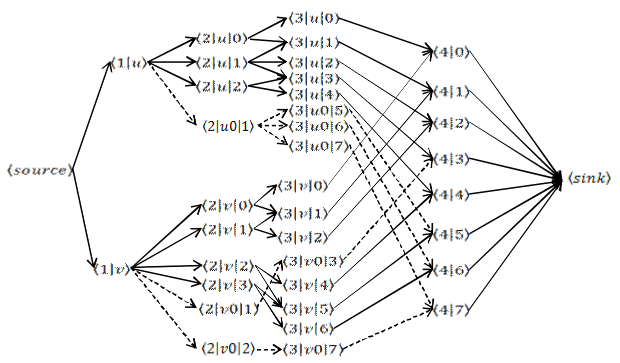

Figure 4 Pfair-graph $G(8)$ for the task system $\Gamma = \{< 0, 3, 5, 8 >, < 0, 2, 3, 4 >\}$

to this node corresponds to the global processor demand of the task within the interval $[0, L)$.
V_2 contains one node for each sub-task, one node for each before release period (for late released tasks) , and one node for each idle periods within $[0, L]$. Each of these nodes is linked to the node of V_1 corresponding to its parent task.
V_3 contains one node for each before release time unit, one node for each task and for each time unit between release and deadline of each instance, and one node for each time unit of each idle period. Then each before release node of V_2 is linked to each before release node of V_3, each sub-task node is linked to the node corresponding to time units located within its feasibility window, and each idle period node of V_2 is linked to each idle period node of V_3 corresponding to time units located in the corresponding idle period.
Finally, V_4 contains one node for each time unit within the time interval $[0, L)$, which is linked to each node of V_3 corresponding to the same time unit.
In order to prove the theorem, we first establish the following lemma.

Lemma 1. *If the Pfair-graph $G(L)$ has an integral flow of size $\sum_{\tau_i \in \Gamma} CTR_{\tau_i}(L)$, then Γ has a Pfair schedule on $[0, L)$.*

Proof - Let us assume that such an integral flow exists. We first define a schedule SG deduced from the Pfair -graph as:

Definition 4. *Let f be an integral flow of size $\sum_{i=1}^{n} CTR_{\tau_i}(L)$ of G, we define SG as follows. For $\tau_i \in \Gamma$, $t \in \mathbb{N}$,*

$$SG(\tau_i, t) = \begin{cases} 1 \; if \; t \in [0, L) \wedge (\exists j \in [0, CTR_i(L)) :: \\ \quad f((< 2, \tau_i^j >, < 3, \tau_i, t >, 1)) = 1 \\ \\ 0 \qquad\qquad\qquad\qquad otherwise \end{cases}$$

We show that SG is Pfair over the time interval $[0, L)$. The size of the flow is $\sum_{i=1}^{n} CTR_{\tau_i}(L)$ thus each node of V_1 is filled to capacity, i.e. the flow carried by each link from source to $< 1, \tau_i >$ is equal to $CTR_{\tau_i}(L)$. Each node in V_1 has exactly $CTR_{\tau_i}(L)$ outgoing edges of capacity 1, the other outgoing edges have a null capacity thus they receive a flow equal to 0. Thus each node $< 2, \tau_i^j >$ of V_2 receives a flow equal to 1. Then, each node $< 2, \tau_i^j >$ has one single outgoing edge which carries a flow equal to 1, the other outgoing edges carry a flow equal to 0. Now, each node of V_3 has exactly one outgoing edge. This edge carries a flow equal to the flow carried by its incoming edge. In the same way, the nodes in V_4 have one single outgoing edge, which carries the cumulated flow carried by their incoming edges. Since the capacity of this outgoing edge is m, at most m incoming edges carry a flow equal to 1. Thus at most m sub-tasks are processed. Now, the potential exists for a task τ_i to get scheduled twice at the same time (if $r_i^j = d_i^{j-1} - 1$). But since the edge from $< 3, \tau_i, t >$ to $< 4, t >$ has a capacity equal to 1, this situation is avoided: if this edge carries a flow equal to 1, only one incoming edge carries a non-null flow. Thus two different processed sub-tasks cannot belong to the same task. Thus at any time t in $[0, L)$, there exists at most m tasks such that $SG(\tau_i, t) = 1$. Furthermore, $f((< 2, \tau_i^j >, < 3, \tau_i, t >, 1)) = 1$ implies that $r_i^j \leq t < d_i^j$ thus, each processed sub-task is processed in its feasibility window. Finally, each sub-task is effectively processed. Indeed, there exists $\sum_{i=1}^{n} CTR_{\tau_i}(L)$ sub-tasks in $[0, L)$. Following the definition of SG, the number of processed sub-tasks is equal to the global incoming flow of vertices of V_3. And the global input flow of V_3 is constant, equal to $\sum_{i=1}^{n} CTR_{\tau_i}(L)$ by definition of the flow. Thus each sub-task is processed. The schedule SG is thus Pfair on $[0, L)$. □

We now prove the existence of an integer flow of size $\sum_{i=1}^{n} CTR_{\tau_i}(L)$. We use the following flow assignments:

Definition 5. *Let f be the flow defined as:*

- $f((< source >, < 1, \tau_i >, CTR_{\tau_i}(L))) = CTR_{\tau_i}(L)$
- ▷ $f((< 1, \tau_i >, < 2, \tau_{i0}, 0 >, 0)) = 0$
  ▷ $f((< 1, \tau_i >, < 2, \tau_i^j >, 1)) = 1$
  ▷ $f((< 1, \tau_i >, < 2, \tau_{i0}, j >, 0)) = 0$
- ▷ $f((< 2, \tau_{i0}, 0 >, < 3, \tau_{i0}, t >, 0)) = 0$
  ▷$f((< 2, \tau_{i0}, j >, < 3, \tau_{i0}, t >, 0)) = 0$
  ▷
  ⋆ $f((< 2, \tau_i^j >, < 3, \tau_i, r_i^j >, 1)) =$
  $CH_i - (j - r_i^j.CH_i)$
  ⋆ $f((< 2, \tau_i^j >, < 3, \tau_i, d_i^j - 1 >, 1)) =$
  $(J + 1) - r_i^{j+1}.CH_i$ *if* $d_i^j - 1 = r_i^{j+1}$
  ⋆ *Otherwise* $f((< 2, \tau_i^j >, < 3, \tau_i, t >, 1)) = CH_i$
- ▷ $f((< 3, \tau_{i0}, t >, < 4, t >, 0)) = 0$
  ▷$f((< 3, \tau_i, t >, < 4, t >, 1)) = CH_i$
- $f((< 4, t >, < sink >, m)) = \sum_{\substack{\tau_i \in \Gamma s.t. \\ r_i + k.P_i \leq t < r_i + k.P_i + D_i}} CH_i$

Lemma 2. *f is a flow of size $\sum_{\tau_i \in \Gamma} CTR_{\tau_i}(L)$ of $G(L)$*

We first prove that the capacity constraints are met. Edges in E_0, E_1 are filled to capacity, and edges in E_3 carry flows either equal to 0 or to the density CH_i which is less than or equal to 1, thus capacity constraints are met. If an edge in E_4 is considered, it carries a flow $\sum_{\substack{\tau_i \in \Gamma \text{s.t.} \\ r_i + k.P_i \leq t < r_i + k.P_i + D_i}} CH_i \leq \sum_{\tau_i \in \Gamma} CH_i$, now by assumption we have $\sum_{\tau_i \in \Gamma} CH_i \leq m$, so the capacity constraint is met. Finally, for edges in E_2, we must prove that $CH_i - (j - r_i^j.CH_i) \leq 1$ and $(j+1) - r_i^{j+1}.CH_i \leq 1$ if $d_i^j - 1 = r_i^{j+1}$. We have $r_i^j = \lfloor \frac{j}{CH_i} \rfloor$ thus $\frac{j}{CH_i} - 1 < r_i^j \leq \frac{j}{CH_i}$ thu s$-CH_i < r_i^j.CH_i - j \leq 0$ thus $0 < CH_i - (j - r_i^j.CH_i) \leq CH_i \leq 1$. We prove that $(j+1) - r_i^{j+1}.CH_i \leq 1$ using similar arguments. Thus capacity constraints are all met. We must then show that the flow is preserved at every inner vertex. For null capacity nodes, the flow is clearly preserved. For any node $< 1, \tau_i >$ of V_1, the incoming flow is $CTR_{\tau_i}(L)$, and the outgoing flow is equal to the number of sub-tasks since edges are filled to capacity, thus the outgoing flow is $CTR_{\tau_i}(L)$ too. Each vertex $< 2, \tau_i^j >$ has an incoming flow of 1. Each vertex $< 2, \tau_i^j >$ has $d_i^j - r_i^j$ outgoing edges. Then the flow out of $< 2, \tau_i j >$ is, if $d_i^j - 1 = r_i^{j+1}$, $CH_i - (j - r_i^j.CH_i) + CH_i.(d_i^j - r_i^j - 2) + (j+1) - r_i^{j+1}.CH_i$ which simplifies to 1. Otherwise, we have $d_i^j - 1 \neq r_i^{j+1}$ thus $\lceil \frac{J+1}{CH_i} \rceil - 1 \neq \lfloor \frac{j+1}{CH_i} \rfloor$ which means that $\frac{j+1}{CH_i}$ is integral thus $d_i^j = \frac{j+1}{CH_i}$. Now, the flow out is $CH_i - (j - r_i^j.CH_i) + CH_i.(d_i^j - r_i^j - 1)$ which then simplifies to 1. There is only one outgoing edge leaving any vertex $< 3, \tau_i, t >$ of V_3, which carries a flow equal to CH_i. If $d_i^j - 1 = r_i^{j+1}$, then there are two incoming edges which carry a flow of size $(j+1) - r_i^{j+1}.CH_i + CH_i - ((j+1) - r_i^{j+1}.CH_i) = CH_i$. Otherwise there is only one incoming edge which carries a flow equal to CH_i. We consider finally a vertex $< 4, t >$ of V_4. Its incoming edges with non zero capacity are edges $(< 3, \tau_i, t >, < 4, t >, 1)$ with $r_i + k.P_i \leq t < r_i + k.P_i + D_i$. Thus the incoming flow is $\sum_{\substack{\tau_i \in \Gamma \text{s.t.} \\ r_i + k.P_i \leq t < r_i + k.P_i + D_i}} CH_i$, which is thus equal by definition to the flow of the unique outgoing edge. Thus, we proved that the flow is preserved at any inner node. Thus f is a flow of size $\sum_{\tau_i \in \Gamma} CTR_{\tau_i}(L)$. □
We can now complete the proof of theorem 2. Lemma 2 implies the existence of a fractional flow of size $\sum_{\tau_i \in \Gamma} CTR_{\tau_i}(L)$ for the Pfair-graph $G(L) = (V, E)$. Since capacities are integral, this implies the existence of an integral flow of size $\sum_{\tau_i \in \Gamma} CTR_{\tau_i}(L)$ in $G(L)$ Jr and Fulkerson (1962). Then Lemma 1 proves that a Pfair schedule can be constructed. This proves theorem 2. □

Then we extend the algorithm PF to periodic task sets ($r_i \geq 0, D_i \leq T_i$). Here, a task can be *Urgent*, *Tnegru* or *Contending* if $t \in [k.T_i + r_i, k.T_i + r_i + D_i)$ and is *Idle* if $t < r_i$ or $t \in [k.T_i + r_i + D_i, (k+1).T_i)$. The extension is then straightforward: at each time t, *Urgent* tasks are scheduled, *Contending* tasks are sorted and the first of them are allocated to the remaining processors.

Number of processors	2	3	4	5	6	%
Number of systems	1000	1000	1000	1000	1000	
$0 \leq t_0 \leq max(r_i)$	0	0	0	0	0	0
$t_0 = max(r_i)$	852	741	721	689	705	74.16
$max(r_i) < t_0 \leq max(r_i) + P$	148	259	279	311	295	25.84
$t_0 > max(r_i) + P$	0	0	0	0	0	0

Table 1 Simulation results about cyclicity - t_0 is the date of the beginning of the steady state.

4 Simulation results

For periodic task sets, we have proposed a sufficient feasibility condition: there exists a valid Pfair schedule on m processors for every task set such that $CH = \sum_{i=1}^{i=n} \frac{C_i}{D_i} \leq m$. The next point of interest is to determine whether this condition is efficient. We thus investigate the soundness of our bound. For that purpose, we have carried out some simulations. We have first implemented a task set simulator and a scheduler based on our extension of PF. We have then generated task sets with different values of either $U = \sum_{i=1}^{i=n} \frac{C_i}{T_i}$ or $CH = \sum_{i=1}^{i=n} \frac{C_i}{D_i}$. And we estimate the ratio of feasible sets among them. We present hereafter the different steps of simulations.

4.1 The task set generator

We must experiment PF on a significant number of task sets. Thus, we need to generate different samples of task sets. The generator generates randomly task parameters. In order to limit the scheduling step, we generate periods according to Goossens methodology Macq and Goossens (2001), which permits to get a bound for the hyperperiod (the LCM of the task periods). For our simulations the upper bound of hyperperiods is set to 210. Offsets, constrained deadlines and WCETare chosen uniformly within respectively the intervals $[0, T_i]$, $[1, T_i]$ and $[1, D_i - 1]$.

4.2 The scheduler

The scheduler produces the PF schedule according to our adapted PF policy, over a given time interval. An important point was to determine the time interval that must be used. For synchronous task sets, the length of the schedule generated is one hyperperiod. For asynchronous task sets, we don't have enough results on the cyclicity in this context. For uniprocessor systems, it has been shown in

Tasks set Model		$U \leq m$	$CH \leq m$	$CH > m$
$r_i = 0$	$D_i = T_i$	$N = 100\%$		
$r_i = 0$	$D_i \leq T_i$	$0 < N < 100\%$	$N = 100\%$	$0 < N < 100\%$
$r_i \geq 0$	$D_i = T_i$	$N = 100\%$		
$r_i \geq 0$	$D_i \leq T_i$	$0 < N < 100\%$	$N = 100\%$	$0 < N < 100\%$

Table 2 Global results of simulations - N is the ratio of Pfair-feasible task sets within the considered sample

Choquet-Geniet and Grolleau (2004) that the steady state starts no later than $max\{r_i\}_{i=1..n} + T$. For multiprocessor systems and for any preemptive and fixed priorities scheduling, in Cucu and Goossens (2007), the authors show that the steady state begins before at S_n where T denotes the hyperperiod and S_i is defined inductively by:

- $S_1 = r_1$
- $S_i = max\left\{r_i, r_i + \left\lceil \frac{S_{i-1} - r_i}{T_i} \right\rceil .T_i\right\}$ $\forall i \in \{2, 3, ...n\}$.

In Choquet-Geniet and Malo (2009), an algorithmic characterization of the beginning of the steady state is investigated for work-conserving strategies. At our knowledge, there is no result on the date of beginning of the steady state for Pfair scheduling strategies. We thus have carried out simulations until detection of the steady state. In our context, for all simulations, the steady state has started before $max\{r_i\}_{i=1...n} + T$. Thus, all simulations have been carried out over a time interval included in $[0, max\{r_i\}_{i=1...n} + 2 * T)$. Table 1 summarizes our simulation results on cyclicity for a Pfair scheduling strategy.

4.3 Simulations

We have considered 8 cases. The results we got are summarized in table 2, missing values correspond to not investigated cases. Task sets can be synchronous or asynchronous, with implicit or constrained deadlines. They are characterized by either $U \leq m$ or $CH \leq m$ or $CH > m$. For each case, we have generated a sample of 5000 tasks sets for simulations. As expected, we find a Pfair feasibility rate of 100% for systems with implicit deadlines: for synchronous systems, it corresponds to Baruahs theorem (theorem 1), and for asynchronous systems, it comes from our result (theorem 2). For the other cases, we conclude that:

1. For constrained task sets, $U \leq m$ is no more a sufficient condition, since there exists Pfair unfeasible task sets with a utilization factor less than m.
2. For constrained task sets, $CH \leq m$ is not a necessary condition since there exist Pfair feasible task sets with $CH > m$.

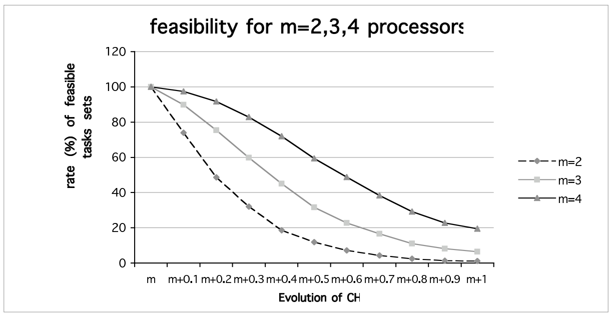

Figure 5 Pfair feasibility rate according to CH for 2,3,4 processors

Then we refined our simulations in order to determine the incidence of U or CH on Pfair feasibility. On figure 5 , we have considered systems of 2,3 or 4 processors. For each case, we generate samples for different values of CH between m and $m+1$, namely $CH = m + \frac{k}{10}$ $(0 \leq k \leq 10)$. For each value of CH, we again determine the ratio of Pfair feasible task sets. We can see that this rate decreases rather quickly when CH increases. If CH remains close to m, the rate of valid system remains high, but the slope of the curve is high and consequently the rate becomes very small if CH approaches $m+1$. We can conclude from these results that our bound is rather good in the sense that only few systems rejected by our test are in fact Pfair feasible.
We also investigate the correlation between U and the Pfair feasibility (see figure 6 and 7). We consider systems with 3 or 4 processors, and synchronous task systems with constraint deadlines. We see that if U is close to m, then quite no systems are Pfair feasible. But if U is less then $\frac{m}{2}$, we have 100% of Pfair feasible systems. Further investigations must be done here.

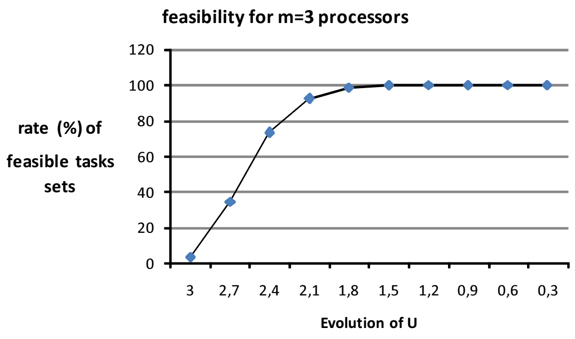

Figure 6 Pfair feasibility according to U for 3 processors for synchronous task systems with constraint deadlines

5 Conclusions

We have extended Pfairness to any set of periodic independent task set. We have considered as well late released tasks as constraint deadlines. We have proposed a sufficient condition and given an adapted version of PF. We proved the

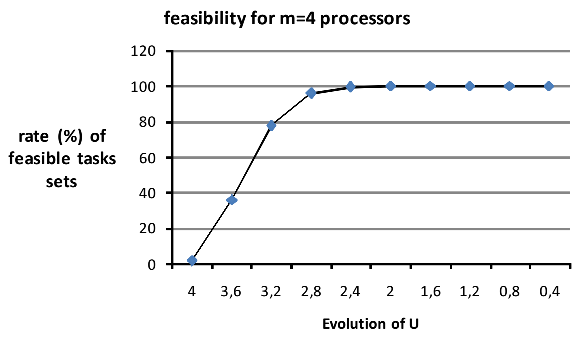

Figure 7 Pfair feasibility according to U for 4 processors

existence of a Pfair schedule if $CH = \sum_{i=1}^{i=n} \frac{C_i}{D_i} \leq m$. Then we have presented simulation results, which illustrate the soundness of our sufficient conditions. These simulations show that if CH increases from m, the rate of Pfair feasible systems decreases quickly. We also present some results about the incidence of the utilisation factor $U = \sum_{i=1}^{i=n} \frac{C_i}{T_i}$. We speculate about the existence of a lower bound for U, which can be used for any periodic task set.
Future investigations will concentrate on the one hand on refinements of our sufficient condition and of the bound for either CH or U, and on the other hand on the cyclicity properties of Pfair schedules for systems with late released tasks, and on the date of the beginning of the steady state.

References

Anderson, J., Block, A., and Srinivasan, A. (2000). Pfair scheduling : Beyond periodic task sytems. In *Proceedings of the* 12^{th} *Euromicro Conference on Real-Time Systems*, pages 35–43. Chapman and Hall.

Anderson, J., Holman, P., and Srinivasan, A. (2004). Fair scheduling of real time tasks on multiprocessors. *Handbook of scheduling : Algorithms, Models and Performance analysis*, pages 31.1–31.21.

A.Srinivasan, Holman, P., Anderson, J., and Baruah, S. (2003). The case for fair multiprocessor scheduling. *Parallel and Distributed Processing Symposium, International*, 0.

Baruah, S., Cohen, N., Plaxton, C., and Varvel, D. (1996). Proportionate progress : a notion of fairness in resource allocation. *Algorithmica*, 15:600–625.

Baruah, S., Gehrke, J., and Plaxton, C. (1995). Fast scheduling of periodic tasks on multiple resources. In *Proceedings of the* 9^{th} *International Parallel Processing Symposium*, pages 280–288.

Choquet-Geniet, A. and Grolleau, E. (2004). Minimal schedulability interval for real time systems of periodic tasks with offsets. *Theoretical of Computer Science*, pages 117–134.

Choquet-Geniet, A. and Malo, S. (2009). Finding cyclicity behavior in multiprocessor scheduling. Technical report, LISI - ENSMA and University of Poitiers.

Cucu, L. and Goossens, J. (2007). Feasibility intervals for multiprocessor fixed-priority scheduling of arbitrary deadline periodic systems. In *Design, Automation and Test in Europe*, pages 1635–1640. IEEE Computer Society.

Dertouzos, M. and Mok, A. (1989). Multiprocessor scheduling in hard real-time environment. *IEEE transactions on sofware Engineering*, 15(12):1497–1506.

Devi, U. and Anderson, J. (2005). Desynchronized pfair scheduling on multiprocessors. *Parallel and Distributed Processing Symposium, International*, 1.

Jr, L. F. and Fulkerson, D. (1962). *Flows in networks*. Princeton University Press.

Macq, C. and Goossens, J. (2001). Limitation of the hyper-period in real-time periodic task set generation. In Teknea, editor, *Proceedings of the 9th international conference on real-time systems*, pages 133–148, Paris France. ISBN 2-87717-078-0.

Mok, A. and Dertouzos, M. (1978). Multi processor scheduling in a hard real-time environment. In *Proc. of* 7^{th} *Texas Conference on Computer Systems*.

Ramamurthy, S. (2002). Scheduling periodic hard real-time tasks with arbitrary deadlines on multiprocessors. In *IEEE Real-Time Systems Symposium*.