The scheduling algorithms PF et PD^2 are monotonous

Annie Choquet-Geniet

25 juin 2010

1 PFair Scheduling

For any real x, $\lfloor x \rfloor$ denotes the greatest integer less than or equal to x and $\lceil x \rceil$ the smallest integer greater than or equal to x.
We consider a platform composed of m identical processors and applications composed of n periodic tasks with implicit deadlines $\tau_i(C_i, T_i)$. C_i is the worst case execution time of the task (WCET) and T_i is its period. We assume that the tasks are submitted to hard deadlines, and that they can share critical resources (shared variables, memory segments, ...). A task consists of an infinity of instances (or jobs) activated at times $k.T_i$ ($k \in \mathbb{N}$). At each new activation, the previous instance must have completed exectution. We assume that parallelism is forbidden: a task cannot be processed on several processors at the same time. We finally assume that the temporal parameters are known and deterministic. We denote T the hyperperiod of the system defined by $T = LCM\{T_1, T_2, \dots, T_n\}$.
Formally, a schedule is an application $S : \mathbb{N} \times \{1, \dots, n\} \to \{0, 1\}$ such that $\sum_{i=1}^{n} S(t, i) \leq m$. Intuitively, $S(t, i) = 1$ means that the task τ_i is processed within the time interval $[t, t+1)$ on one processor. In the sequel, we use the notation $S(t, i) = S_i(t)$.
PFair strategies have been developed for multiprocessor systems. They are optimal for independent task sets, in multiprocessor environment. The basic idea is that the tasks are processed at regular rate, equal to their utilization factor ($u_i = \frac{C_i}{T_i}$). This means that at time t, the task τ_i must have been processed for $\frac{C_i}{T_i}.t$ time units. Now, the number of already processed time units must be integer, it is thus approximated either by $\lfloor u_i.t \rfloor$ or by $\lceil u_i.t \rceil$. Formally, we have the following definition:

Definition 1. *A schedule S is PFair if and only if*

$$\forall t \in \mathbb{N}, \forall i \in \{1, \dots, n\}, -1 < u_i.t - \sum_{j=0}^{t-1} S_i(j) < 1.$$

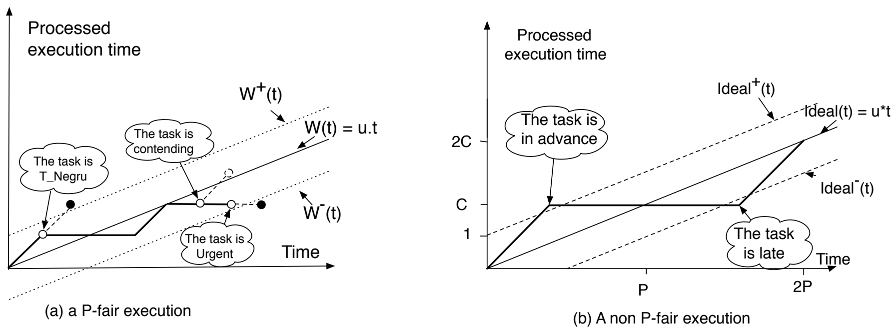

FIG. 1 – PFair and non PFair executions - Different status of a task.

The figure 1 illustrates the notion of PFairness. For each task, the curve $C_i(t)$ which represents the already processed time units must remain between the limit lines of equation $y = \frac{C_i}{P_i}.t \pm 1$.
At any time t, if the point $C_i(t)$ is above the ideal line, the task is said to be **ahead**, if it is on the ideal line, the task is said to be **punctual** and if it is below the ideal line, the tasks is said to be **behind**. In order to get a PFair schedule, at any time $t \in \mathbb{N}$, the task set $\{\tau_1, \tau_2, \ldots, \tau_n\}$ is partitionned into three subsets (see figure 1):

- **Urgent** tasks are those which are behind, and, which, if they were not processed at time t, would be below the lower limit line at time $t + 1$.
- **Tnegru** tasks are those which are ahead, and which, if they were processed at time t would be above the upper limit line at time $t + 1$.
- **Contending** tasks are those of which neither the execution nor the idelness at time t would violate the PFairness condition.

All Urgent tasks are processed.
Contending tasks are sorted, and the $m- \mid Urgent \mid$ first ones are scheduled[1].
Two PFair algorithms have been proposed in the litterature (PF and PD^2 [2, 1]). They differ from each other in the way they select processed Contending tasks, thus in the way they sort them. Both are optimal.

1.1 The algorithm PF

We first consider the algorithm PF. We use the initial definitions given in [2]. We introduce the notion of characterisitic string. The characteristic string α of a task τ_i is the infinite string $[\alpha_i(t)]_{t\in\mathbb{N}}$ over {-, 0, +} such that

$$\alpha_i(t) = sign(u_i \times (t + 1) - \lfloor u \times t \rfloor - 1)$$

The characteristic substring of τ_i at time t is defined by

$$\alpha(i, t) = \alpha_i(t + 1) \ldots \alpha_i(t')$$

[1] $\mid A \mid$ denotes the cardinality of A

where t' is the first time after t such that $\alpha_i(t') = 0$. The algorithm PF is depicted as follows [2] :

1. Urgent tasks are scheduled (let k be their number)
2. Contending tasks are sorted, using the lexicographical order on their characteristic substrings (with $- \prec 0 \prec +$).
3. The $(m-k)$ first Contending tasks are scheduled. Tie-breaks are resolved by a deterministic rule.

1.2 The algorithm PD^2 [1]

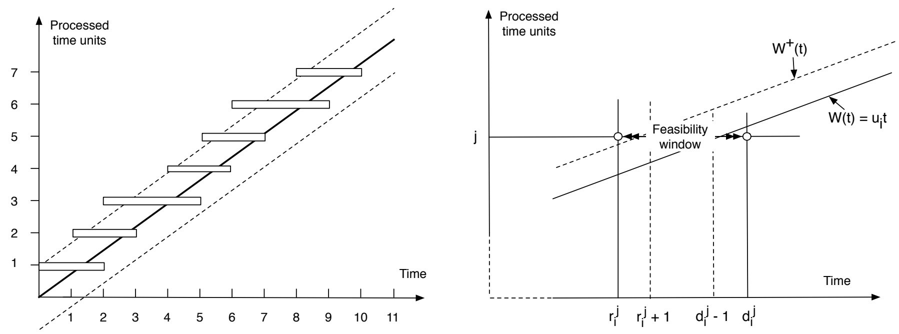

FIG. 2 – Feasibility windows

Each task τ_i is decomposed into an infinity of unitary subtasks τ_i^j $(j \geq 1)$. The j^{th} subtask must then be scheduled within it feasibility window I_i^j delimited by $r_i^j = \lfloor \frac{j-1}{u_i} \rfloor$ and $d_i^j = \lceil \frac{j}{u_i} \rceil$ (see figure 2). We denote $\mid I_i^j \mid$ the size of the window I_i^j: $\mid I_i^j \mid = d_i^j - r_i^j$. For each subtask τ_i^j, the successor bit is defined by

$$b_i^j = d_i^j - r_i^{j+1} = \lceil \frac{j}{u_i} \rceil - \lfloor \frac{j}{u_i} \rfloor$$

It is equal to 1 if the two consecutive windows I_i^j and I_i^{j+1} overlap by one time unit and it is equal to 0 if they are disjoint. Finally, each subtask τ_i^j has a group deadline D_i^j.
If $u_i \geq \frac{1}{2}$ (the task is light), this group deadline is equal to 0: $D_i^j = 0$.
If $u_i < \frac{1}{2}$ (the tasks is heavy), it has been proved that its first feasibility window is of size 2, and the other ones have sizes equal either to 2 or to 3. A group is a sequence of consecutive subtasks $\tau_i^u, \tau_i^{u+1}, \ldots, \tau_i^v$ such that:

- $b_i^u = 1$
- $\mid I_i^k \mid = 2$ and $b_i^k = 1$, $\forall k$ such that $u < k \leq v$
- $\mid I_i^{v+1} \mid = 3$ or $(\mid I_i^{v+1} \mid = 2$ and $b_i^{v+1} = 0)$

In this case, the group deadline is equal to: $\left\{ \begin{array}{l} d_i^v + 1 \ if \ \mid I_i^{v+1} \mid = 3 \\ d_i^v \ if \ b_i^v = 0 \end{array} \right.$

And finally we have $D_i^j = \text{Min}\{t \text{ s.t. } d_i^j \leq t$ and t is a group deadline$\}$. The algorithm PD^2 is then defined as a dynamic priority algorithm: a subtask τ_i^u has priority over a subtask τ_k^v if and only if:

1. $d_i^u < d_k^v$. Its deadline is the nearest.
2. $d_i^u = d_k^v$ and $b_i^u > b_k^v$.
3. $d_i^u = d_k^v$ and $b_i^u = b_k^v = 1$ and $D_i^u > D_k^v$.

2 Monotonous schedules

Monotonous scheduling are such that at any time t, the pending instance of a task cannot have been processed less than the instance pending one hyperperiod later. In the sequel, we use the following notations:

$P_i(t)$ is the pending instance of the task τ_i at time t (the last instance released at or before t).

$RCT_i(t)$ is the remaining computation time for the instance $PI_i(t)$ of the task τ_i.

$\overline{RCT}_i(t)$ corresponds to the elapsed computation time of the instance $PI_i(t)$.

$W_i(t, t')$ is the processed execution time for the task τ_i between time t and time t'.

The monotony property is formally expressed by:

Definition 2. *A schedule is* **monotonous** *iff* $\forall t, \forall i \in 1..n$,

$$RCT_i(t) + \overline{RCT_i}(t+T) \leq C_i$$

This property can equivalently be expressed as:

$$\overline{RCT_i(t)} \geq overline RCT_i(t+T)$$

We consider PF and PD^2 schedules. We assume that ties are resolved by a deterministic rule. E.g., the task with the smallest number will have the higher priority.

Proposition 3. *PF and PD^2 schedules with a deterministic tie-break rule are monotonous.*

Proof

In both proofs, we reason by contradiction. We thus assume that the result doesn't hold. We want to prove that $\forall \tau_i$, $\forall t \geq r_i$, $\overline{RCT}_i(t) \geq \overline{RCT}_i(t+T)$.

We assume the opposite. There is thus a time t_0 which is the first time where at least one task τ_{i_0} verifies $\overline{RCT}_{i_0}(t_0) < \overline{RCT}_{i_0}(t_0 + T)$. We thus have

$$\left\{ \begin{array}{ll} \overline{RCT}_{i_0}(t_0) < \overline{RCT}_{i_0}(t_0 + T) & (1) \\ \overline{RCT}_{i_0}(t_0 - 1) = \overline{RCT}_{i_0}(t_0 - 1 + T) & (2) \end{array} \right.$$

The task τ_{i_0} is thus processed at time $t_0 + T - 1$ but not at time $t_0 - 1$.

1 - We first consider the algorithm PF.

Lemma 4. *The characteristic substrings of τ_i are periodic with period T_i.*

This is a straightforward consequence of the definitions.

We assume that t_0 et τ_{i_0} are defined. We discuss on the status of the task τ_{i_0} at time $t_0 - 1$.

• The task τ_{i_0} cannot be Urgent at time t_0-1, else, it would have been processed.

• If it is Tnegru, this means that $W_{i_0}(0, t_0 - 1) + 1 \geq u_{i_0} \times t_0 + 1$ (should the task be processed, then its execution curve would be above the upper limit line). But in this case, according to the equality (2), we have:
$W_{i_0}(0, t_0 + T - 1) + 1 = W_{i_0}(0, t_0 - 1) + 1 + \frac{T}{T_{i_0}} \times C_{i_0}$ and thus we have $W_{i_0}(0, t_0 + T - 1) + 1 \geq u_{i_0} \times t_0 + 1 + \frac{T}{T_{i_0}} \times C_{i_0} = u_{i_0} \times (t_0 + T) + 1$. Thus τ_{i_0} is also Tnegru at time $t_0 + T - 1$ thus it is processed. This contradicts our assumption.

• The taskτ_{i_0} is thus Contending at time $t_0 - 1$. It is not processed, thus there exist m tasks $\tau_{i_1}, \tau_{i_2}, \ldots, \tau_{i_m}$ which are processed. They are thus either Urgent or Contending.

- If a task τ_{i_j} is Urgent at time $t_0 - 1$, we have $W_{i_j}(0, t_0 - 1) \leq u_{i_j} \times t_0 - 1$ (shouldn't the task be processed, then its execution curve would be located below the lower limit line). Besides, from the definition of t_0, we have $\overline{RCT}_{i_j}(t_0-1) \geq \overline{RCT}_{i_j}(t_0+T-1)$ thus we have $W_{i_j}(0, t_0-1)+\frac{T}{T_{i_j}} \times C_{i_j} \geq W_{i_j}(0, t_0+T-1)$. It follows that $W_{i_j}(0, t_0+T-1) \leq u_{i_j} \times t_0 - 1 + \frac{T}{T_{i_j}} \times C_{i_j} = u_{i_j} \times (t_0 + T) - 1$. The task τ_{i_j} is thus also Urgent at time $t_0 + T - 1$ and it is thus processed.

- If a task τ_{i_j} is Contending, it has priority over τ_{i_0} because:
  ⋄ either its characteristic substring is smaller (for the lexicographical order) than the substring of τ_{i_0}: $\alpha(i_j, t_0 - 1) \prec \alpha(i_0, t_0 - 1)$;
  ⋄ either both substrings are equal, and the tie-break rule gives τ_{i_j} the higher priority.
  At time t_0+T-1, we have $\overline{RCT}_{i_j}(t_0-1) \geq \overline{RCT}_{i_j}(t_0+T-1)$, thus either τ_{i_j} is Urgent, and it is thus processed, either it is Contending. Since characteristic substrings are periodic (lemma 4), we have $\alpha(i_j, t_0 + T - 1) \prec \alpha(i_0, t_0 + T - 1)$ or $\alpha(i_j, t_0 + T - 1) = \alpha(i_0, t_0 + T - 1)$ and τ_{i_j} has priority over τ_{i_0} (because of the determinism of the tie-break rule). Thus τ_{i_j} has again priority over τ_{i_0} at time $t_0 + T - 1$.

It follows that τ_{i_0} cannot be processed at time t_0+T-1. This contradicts our assumption. Thus we have $\forall \tau_i$, $\forall t \geq r_i$, $\overline{RCT}_i(t) \geq \overline{RCT}_i(t+T)$. The algorithm PF is thus monotonous.

2 - We consider now the algorithm PD^2. We recall that τ_i^j denotes the i^{th} unitary subtask of τ_i, r_i^j denotes its pseudo release ant d_i^j its pseudo deadline.
We consider a time t and a task τ_i. We denote $j_i(t)$ the number of the next subtask of τ_i to be processed. The subtask number $j_i(t)-1$ has already been processed, but not the subtask number $j_i(t)$. We thus have $j_i(t) = W_i(0,t)+1$.
We assume that t_0 and τ_{i_0} are defined as previously described.
We have $\overline{RCT}_{i_0}(t_0-1) = \overline{RCT}_{i_0}(t_0+T-1)$.
It follows that $W_{i_0}(0,t_0-1) + \frac{T}{T_{i_0}} \times C_{i_0} = W_{i_0}(0,t_0+T-1)$ and thus:

(i) $j_{i_0}(t_0+T-1) = j_{i_0} + \frac{T}{T_{i_0}} \times C_{i_0}$
(ii) $d_{i_0}^{j_0(t_0+T-1)} = d_{i_0}^{j_0(t_0-1)} + T$
(iii) $r_{i_0}^{j_0(t_0+T-1)} = r_{i_0}^{j_0(t_0-1)} + T$
(iv) $b_{i_0}^{j_0(t_0+T-1)} = b_{i_0}^{j_0(t_0-1)}$
(v) The group deadline of $\tau_{i_0}^{j_0(t_0+T-1)}$ is equal to the deadline of $\tau_{i_0}^{j_0(t_0-1)} + T$: $D_{i_0}^{j_0(t_0+T-1)} = D_{i_0}^{j_0(t_0-1)} + T$

Moreover, we have $\overline{RCT}_{i_0}(t_0) < \overline{RCT}_{i_0}(t_0+T)$ thus

$$\begin{cases} \tau_{i_0}^{j_{i_0}(t_0-1)} \textit{ is not processed at time } t_0-1. \\ \tau_{i_0}^{j_{i_0}(t_0+T-1)} \textit{ is processed at time } t_0+T-1 \end{cases}$$

Now $\tau_{i_0}^{j_{i_0}(t_0-1)}$ is not processed at time t_0-1 if:

- $t_0 - 1 < r_{i_0}^{j_{i_0}(t_0-1)}$.
  But in this case, we would have $t_0+T-1 < r_{i_0}^{j_{i_0}(t_0-1)} + T = r_{i_0}^{j_{i_0}(t_0+T-1)}$ and thus $\tau_{i_0}^{j_{i_0}}(t_0+T-1)$ couldn't have been processed at time t_0+T-1.

- there are m subtasks with higher priorities. Let $\tau_{i_1}, \tau_{i_2}, \ldots, \tau_{i_m}$ be their parent tasks (τ_i is the parent task of the subtasks τ_i^j).
  Following the definition of t_0, we have $\overline{RCT}_{i_j}(t_0-1) \geq \overline{RCT}_{i_j}(t_0+T-1)$.
  Therefore, $j_{i_k}(t_0+T-1) \leq j_{i_k}(t_0-1) + \frac{T}{T_{i_k}} \times C_{i_k}$, and thus
  (A) $d_{i_k}^{j_{i_k}(t_0+T-1)} \leq d_{i_k}^{j_{i_k}(t_0-1)} + T$.
  A subtask $\tau_{i_k}^{j_k(t_0-1)}$ has priority over $\tau_{i_0}^{j_{i_0}(t_0-1)}$ if:
  - $d_{i_k}^{j_{i_k}(t_0-1)} < d_{i_0}^{j_0(t_0-1)}$.
    In this case, following (A) and (ii), $d_{i_k}^{j_{i_k}(t_0+T-1)} \leq d_{i_k}^{j_{i_k}(t_0-1)} + T < d_{i_0}^{j_{i_0}(t_0-1)} + T = d_{i_0}^{j_{i_0}(t_0+T-1)}$. Thus $\tau_{i_k}^{j_{i_k}(t_0+T-1)}$ has also priority over $\tau_{i_0}^{j_{i_0}(t_0+T-1)}$.
  - $d_{i_k}^{j_{i_k}(t_0-1)} = d_{i_0}^{j_0(t_0-1)}$and $b_{i_k}^{j_{i_k}(t_0-1)} > b_{i_0}^{j_0(t_0-1)}$.

In this case,
• either $j_{i_k}(t_0+T-1) = j_{i_k}(t_0-1) + \frac{T}{T_{i_k}} \times C_{i_k}$.
Then we have $d_{i_j}^{j_j(t_0+T-1)} = d_{i_0}^{j_0(t_0+T-1)}$. We also have
$b_{i_k}^{j_{i_k}(t_0+T-1)} = b_{i_k}^{j_{i_k}(t_0-1)} > b_{i_0}^{j_0(t_0-1)} = b_{i_0}^{j_0(t_0+T-1)}$.
Thus $\tau_{i_k}^{j_{i_k}(t_0+T-1)}$ has priority over $\tau_{i_0}^{j_{i_0}(t_0+T-1)}$.
• either $j_{i_k}(t_0+T-1) < j_{i_k}(t_0-1) + \frac{T}{T_{i_k}} \times C_{i_k}$.
In this case, $d_{i_k}^{j_{i_k}(t_0+T-1)} < d_{i_0}^{j_{i_0}(t_0+T-1)}$. Thus $\tau_{i_k}^{j_{i_k}(t_0+T-1)}$ has priority over $\tau_{i_0}^{j_{i_k}(t_0+T-1)}$.

– $d_{i_k}^{j_{i_k}(t_0-1)} = d_{i_0}^{j_0(t_0-1)}$, $b_{i_k}^{j_{i_k}(t_0-1)} = b_{i_0}^{j_0(t_0-1)} = 1$ et $D_{i_k}^{j_{i_k}(t_0-1)} > D_{i_0}^{j_0(t_0-1)}$.
Then,
• either $j_{i_k}(t_0+T-1) = j_{i_k}(t_0-1) + \frac{T}{T_{i_k}} \times C_{i_k}$.

In this case, $d_{i_k}^{j_{i_k}(t_0+T-1)} = d_{i_0}^{j_{i_0}(t_0+T-1)}$ and $b_{i_k}^{j_{i_k}(t_0+T-1)} = b_{i_k}^{j_{i_k}(t_0-1)} = b_{i_0}^{j_{i_0}(t_0-1)} = b_{i_0}^{j_{i_0}(t_0+T-1)} = 1$.
Besides we have $D_{i_k}^{j_{i_k}(t_0+T-1)} = D_{i_k}^{j_{i_k}(t_0-1)} + T > D_{i_0}^{j_{i_0}(t_0-1)} + T = D_{i_0}^{j_{i_0}(t_0+T-1)}$.
It follows that $\tau_{i_k}^{j_{i_k}(t_0+T-1)}$ has also priority over $\tau_{i_0}^{j_{i_0}(t_0+T-1)}$.
• either $j_{i_k}(t_0+T-1) < j_{i_k}(t_0-1) + \frac{T}{T_{i_k}} \times C_{i_k}$. In this case $d_{i_k}^{j_{i_k}(t_0+T-1)} < d_{i_0}^{j_{i_0}(t_0+T-1)}$. Thus $\tau_{i_k}^{j_{i_k}(t_0+T-1)}$ has priority over $\tau_{i_0}^{j_{i_0}(t_0+T-1)}$.

– $d_{i_k}^{j_{i_k}(t_0-1)} = d_{i_0}^{j_0(t_0-1)}$, $b_{i_k}^{j_{i_k}(t_0-1)} = b_{i_0}^{j_0(t_0-1)} = 1$ and $D_{i_k}^{j_{i_k}(t_0-1)} = D_{i_0}^{j_0(t_0-1)}$.
The tie-break rule is then used. Then
• either $j_{i_k}(t_0+T-1) = j_{i_k}(t_0-1) + \frac{T}{T_{i_k}} \times C_{i_k}$, in this case, $d_{i_k}^{j_{i_k}(t_0+T-1)} = d_{i_0}^{j_{i_0}(t_0+T-1)}$ and $b_{i_k}^{j_{i_k}(t_0+T-1)} = b_{i_k}^{j_{i_k}(t_0-1)} = b_{i_0}^{j_{i_0}(t_0-1)} = b_{i_0}^{j_{i_0}(t_0+T-1)} = 1$ and $D_{i_k}^{j_{i_k}(t_0+T-1)} = D_{i_0}^{j_0(t_0+T-1)}$. There is again a tie between the subtasks, and the determnistic rule gives again the higher priority to τ_{i_k}.
• either $j_{i_k}(t_0+T-1) < j_{i_k}(t_0-1) + \frac{T}{T_{i_k}} \times C_{i_k}$.
In this case, $d_{i_k}^{j_{i_k}(t_0+T-1)} < d_{i_0}^{j_{i_0}(t_0+T-1)}$. Thus $\tau_{i_k}^{j_{i_k}(t_0+T-1)}$ has priority over $\tau_{i_0}^{j_{i_0}(t_0+T-1)}$.

Consequently, the m tasks $\tau_{i_1}, \tau_{i_2}, \ldots, \tau_{i_m}$ have priority over τ_{i_0} at time t_0+T-1.

Thus, in any case, τ_{i_0} cannot be processed at time t_0+T-1, what contradicts our assumption. Thus, there exists no couple (t_0, τ_{i_0}) which verifies the properties (1) and (2). The monotony property is thus verified. □

Références

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

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