Integratign PFairness within a modelled based scheduling tool

Annie Choquet-Geniet, Gaëlle Largeteau, Abdoulaye Ouattara

Integrating PFairness within a model based scheduling tool

Annie Choquet-Geniet
LISI
Université de Poitiers and ENSMA
Téléport 2 - 1 avenue Clément Ader
BP 40109
86961 Futuroscope Chasseneuil cedex
France
www.lisi.ensma.fr
annie.geniet@univ-poitiers.fr

Galle Skapin-Largeteau
Xlim-SIC
Université de Poitiers
BP 30179
86962 Futuroscope Chasseneuil cedex
France
http://www.sic.sp2mi.univ-poitiers.fr/
glargeteau@sic.univ-poitiers.fr

Abdoulaye Ouattara
Université Polytechnique
01 BP 1091
Bobo-Dioulasso 01
Burkina Fasso

In this paper, we focus on the use of discrete geometry for the sake of real-time modeling and analysis. We consider multiprocessor context and task sets with offsets, constrained deadlines and critical resources. We want to take regularity criteria into account during the scheduling process. We thus determine the geometrical characterisation of PFair schedules and present the construction steps of a geometric Pfair model. Several uses of this model are then presented: we can select (partially) PFair schedules. We have also defined a PFairness comparison criterion and we use it to choose among a set of feasible schedules the most PFair ones.

discrete geometry, real-time scheduling, modeling, model-checking, PFairness

1. INTRODUCTION

A real-time application must satisfy the time constraints coming from the criticity of certain actions that allow them to interact with the environment of the monitored process. Additional qualitative contraints which insure a good quality of service can also be considered. If the strict time constraints are not satisfied, dysfunctions may occur, which can have unacceptable consequences (e.g. loss of human lives). If the qualitative constraints are not satisfied, the behaviour of the controlled process will have a lower quality.

We are here interested in real-time applications dedicated to the control of processes, with strict time constraints, and regularity requirements in the execution. These requirements are captured by the PFairness property. A real-time application must thus be functionally as well as temporally validated. We focus here on the temporal validation, which relies on an appropriate scheduling policy, that can then be proved to respect all temporal constraints. Two approaches can be considered: the on-line method where a scheduling policy is implemented within the scheduler and the off-line method where a previously computed schedule is stored in a table and then used by a dispatcher. Classically, real-time applications dedicated to process control are modeled as a set of periodic tasks e.g. temperature acquisition in a nuclear station, robot's trajectory computation, processing of information provided by a synchronous link.... Tasks are preemptive, and can share critical resources. We assume that the application runs on a multiprocessor platform composed of m identical processors. We consider only global scheduling (figure 1): tasks can run at any time on any processor, they are never definitively assigned to a given processor. They may start on one processor and resume on another. We also assume that parallelism is forbidden: at

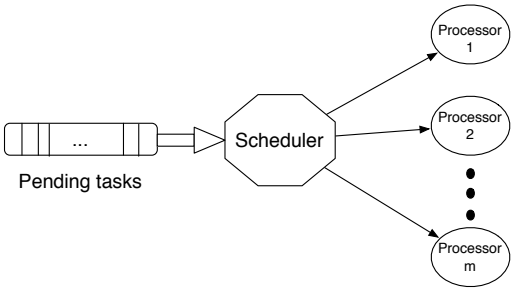

Figure 1: *Global scheduling*

any time, a task runs on at most one processor. In such a context, it has been proven that there exists no optimal on-line strategy (12; 14) where a scheduling algorithm (or strategy) is said to be **optimal** if either it produces a **feasible** (or **valid**) schedule, i.e. a schedule such that all deadlines are met, or there exists no feasible schedule. Furthermore, since critical resources are used, the scheduling problem is NP-hard (1; 17). This motivates the choice of the off-line method. Most off-line strategies relies on branch-and-bound enumeration techniques. These approaches are generally model-driven. The task system is first modeled, then the schedulability analysis relies on exhaustive enumeration techniques. Following, if a valid schedule exists, it will be found. These approaches are thus more powerful than on-line strategies. This can be explained by the fact that scheduling decisions are made according to the instantaneous state of the application for on-line strategies meanwhile they are based on a global knowledge of the application for off-line scheduling. Another benefit of off-line methods is that additional qualitative criteria can be considered. For example, we can find a valid schedule with minimal response times for some tasks, or a valid schedule where some tasks are regularly (or PFairly) processed. Approaches based on finite automata and on Petri nets can be found in the litterature (13; 16). We consider here an approach based on discrete geometry. Such an approach has a lower complexity than the former approaches, as well for the model generation as for the model analysis. In (15), we have defined a discrete geometrical model equivalent to the automata based model presented in (11). In this model, each task is associated with a $2D$-discrete object that collects all the valid states of the task, i.e. all the states *(cumulated processed execution time of the task, time)* belonging to a valid schedule. The shape of this 2D-discrete object depends only on the time characteristics of the task. The execution of the task set is then modeled by a $n+1$-discrete object (where n is the number of tasks). This model is called Concurrency Model and is built using extrusion and intersection of the single task models. Ressource sharing is modeled using a nD-discrete space, extruded following the time direction and then cut out from the concurrency model (see figure 5). We have proved in (15) that there exists a feasible schedule on a m-processor architecture if and only if there exists a m-connected path in the $(n+1)$D-discrete model of the application.
Our aim is to use this model in order to include PFairness properties in the schedule selection step. For independent task systems, PFair strategies have been proposed. Moreover, if tasks are synchronous (they all are first released at the same time) with implicit deadline (deadlines are equal to periods), the algorithms PF, PD and PD^2 (4; 5; 2; 3) are optimal. But in our context, there exists no on-line PFair optimal strategies. We thus want to determine among the valid schedules produced by our off-line methodology which ones are PFair. For that aim, we first present a geometric characterisation of the PFairness, which is used to propose a method for the extraction of (partially) PFair schedules. Then, we propose a method in order to extract as PFair as possible schedules, when no PFair schedule exists. This method relies on PFairness measures.
To our knowledge, there exists not other work on off-line scheduling including PFairness properties. In the litterature, PFair algorithms are meant as on-line strategies. The originlaity of our approach is that we combine fairness and a pre-runtime model based approach, and we can restrict fairness requirements to only some tasks. A parallel study is led, which uses a Petri-net based approach (18).
The paper is organized as follows: in section 2, we introduce basic notions and notations for the real-time scheduling and the discrete modeling of real-time applications. In section 3, we present PFair scheduling and its geometrical modeling. And in section 4, we show how of our model can be used for a qualitative temporal analysis of a real-time application.

2. FUNDAMENTAL DEFINITIONS AND NOTATIONS

2.1. Basic notions of real-time scheduling

We consider multiprocessor systems composed of m identical processors. 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.

2.1.1. The task model

A real-time application consists of a set of n independent periodic tasks $\Gamma = \{\tau_1, \tau_2, \ldots, \tau_n\}$. Each task is submitted to hard temporal constraints. We adopt the classical modeling of tasks. Each periodic task τ is characterized by four temporal parameters as described in figure 2: r is the first release date or offset; C the worst-case execution time; D the relative deadline, which corresponds to the maximal delay allowed between the release and the completion of any instance of the task; and P the period. In the sequel, we denote a task τ by $< r, C, D, P >$. Each task τ consists of an infinite set of instances (or jobs) τ_k, released at times $r + (k-1) \times P$, with k $\in \mathbb{N}^*$. We assume that temporal parameters are known and determinist. We denote H the **hyperperiod** of the system defined as $H = \text{lcm}(P_1, P_2, \ldots, P_n)$.
The processor's **utilisation factor** characterises the processor workload due to the application. It

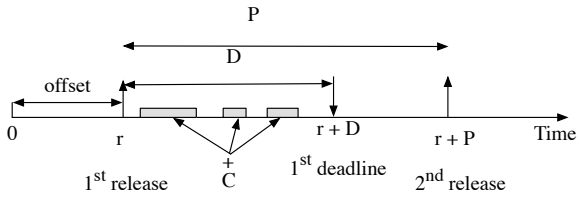

Figure 2: *Temporal modeling of a real time periodic task*

is defined by $U = \sum_{i=1}^{n} \frac{C_i}{P_i}$. If U $>$ m (m being the number of processors), the system is over-loaded and temporal faults cannot be avoided (6). In the sequel, we suppose that the utilisation factor is lower than m.
In the further, **slot** t ($t \in \mathbb{N}$) denotes the time interval $[t, t+1)$. A task is said to be **scheduled at time** t when one processor processes it during slot t.
A **schedule** is defined by $S : \mathbb{N} \times \Gamma \rightarrow \{0, 1\}$ such that $\sum_{i=1}^{n} S(t, \tau_i) \leq m$, $\forall t \in \mathbb{N}$. We have $S(t, \tau) = 1$ $\Leftrightarrow \tau$ is scheduled at time t and if $\sum_{i=1}^{n} S(t, \tau_i) = k < m$ then $(m - k)$ processor idle time units occur at time t, i.e. $(m - k)$ processors remain idle during slot t.
The function S_τ describes the behaviour of a task and is defined by

$$S_\tau(t) = \begin{cases} 1 \text{ if S(t, } \tau\text{) = 1} \\ 0 \text{ else} \end{cases}$$

For any times t and t', and for any task τ, we define $W_\tau(t, t')$ as the **processed execution time** for task τ between time t and time t'.
We have thus $W_\tau(t, t') = \sum_{u=t}^{t'-1} S_\tau(u)$.

2.2. Geometric model for real-time system behaviours

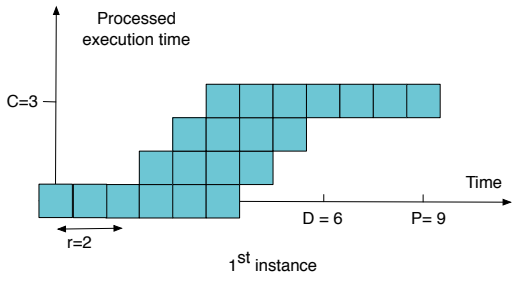

Figure 3: *Geometric model $\Omega(\tau)$ of the task $\tau =< 3, 3, 6, 9 >$*

A task τ is modeled by an object $\Omega(\tau)$ of a 2D space: the (processed execution time, time) space. The shape of this 2D-discrete object depends on the temporal parameters of the task (see figure 3).

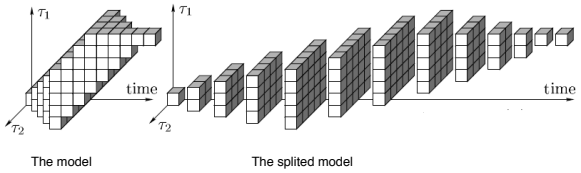

Figure 4: *Geometric model for two tasks*

The model for a task $\tau =< r, C, D, P >$ is formally defined as :

$$\Omega(\tau) = \{(x, t) \in \mathbb{N}^2 s.t. \ \forall t \in \mathbb{N}$$

- $0 \leq t \leq r \Rightarrow x = 0$;
- $r + (k-1)P \leq t \leq r + (k-1)P + D \Rightarrow$ $x \in [(k-1)C, kC]$ *and*

$$(x, t) \in \Omega(\tau) \Rightarrow \begin{cases} (x+1, t+1) \in \Omega(\tau) \ if \ x < kC \\ and \ t < r + (k-1)P + D \\ \\ (x, t+1) \in \Omega(\tau) \ if \\ kC - x < r + (k-1)P + D - t \\ the \ remaining \ time \ until \ the \\ next \ deadline \ is \ greater \ than \ or \\ equal \ to \ the \ remaining \\ processing \ time \ of \ the \ pending \\ instance \end{cases}$$

- $r + (k-1)P + D \leq t \leq r + kP \Rightarrow x = kC\}$

Then to model the complete application $\Gamma = (\tau_i)_{i \in [1,n]}$, we introduce the set

$$\Omega(\Gamma) = \{(x_1, ...x_n, t) \in \mathbb{N}^{n+1}, \forall i \in [1, n], (x_i, t) \in \Omega(\tau_i)\}.$$

For termination reason, we only consider the model over a finite time interval. If tasks are synchronous, cyclicity results (9) show that the construction can be restricted to the time interval $[0, H]$, where H is the hyperperiod. If asynchronous systems are considered, we will restrict ourselves to the time interval $[0, max([r_i]_{i\ 1...n}) + 2 * H]$. Figure 4 presents the model obtained for an application composed of two tasks.
Finaly, concurrency is considered. The final model is deduced from the previous one using extrusion and intersection operations (see figure 5).

Resource sharing between tasks τ_i and τ_j is modeled by a surface in the (x_i, x_j) plan which is then extruded following time direction. The result is finaly cut out from $\Omega(\Gamma)$ to obtain $\Omega(\Gamma, R)$.
We can then deduce the feasibility of the application for a given architecture. If we consider a platform of m identical processors, the existence of a feasible schedule relies on the connectivity properties of the model $\Omega(\Gamma, R)$. Two points $(x_1, \ldots, x_n, t)$ and

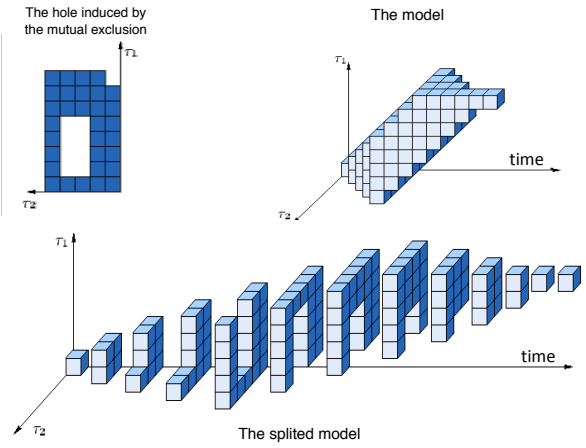

Figure 5: *Integration of the concurrency*

$(x'_1, \ldots, x'_n, t+1)$ are said m-neighbours if $\forall i \in 1 \ldots n \quad \mid x_i - x'_i \mid \leq 1$ and $\sum_{i=1}^{n} \mid x_i - x'_i \mid \leq m$ (see figure 6).
A m-connected path of length T is a set of T m-neighbour points. We finaly have the following result:

Proposition 1 *There exists a feasible schedule of size T for a platform of m identical processors iff there exists a m-connected path (of length T) in the model $\Omega(\Gamma, R)$.*

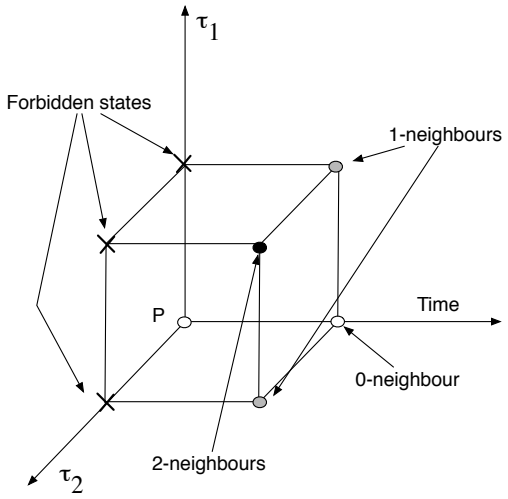

Figure 6: *Neighborhood of the point P*

A complete definition of the geometric model can be found in (15).

3. DISCRETE MODELING OF PFAIRNESS

We want to take a further quality criterion into account: the regularity of execution for (some of) the tasks of the system. This can be useful for some kind of applications, e.g. multimedia applications, to insure a process' behaviour of good quality. This can also be helpful for an efficient aperiodic task management. We first introduce the notion of PFairness. Then, we present the way PFairness is taken into account in our model. In a first time, we consider synchronous tasks with implicit deadlines ($\forall \tau,\ r = 0$ and $D = P$). Then we relax the constraint $r = 0$ and model asynchronous tasks. Next we consider systems with constrained deadlines ($D \leq P$) and finally we take concurrency and critical resource sharing into account.

3.1. PFair scheduling algorithm

PFair scheduling strategies have been proposed in the general multiprocessor context, for which they are very efficient. The basic idea is that each task is processed at "regular rate". This means that at each time t ($t \in \mathbb{N}$), the processed execution time $W_\tau(0,t)$ is proportional to t, with a proportionality coefficient equal to $u = \frac{C}{P}$. But, since the processed execution time at time t must be integer, $u \times t$ is approximated by either $\lfloor u \times t \rfloor$ or $\lceil u \times t \rceil$.
This is formally expressed by the following definition:
A schedule is **PFair** for a task τ iff we have:

$$\forall t \in \mathbb{N}, -1 < u \times t - \sum_{j=0}^{t-1} S_\tau(j) < 1$$

A schedule is PFair if it is PFair for each task τ of Γ. Figure 7 illustrates PFairness. For any task τ, the broken line W_{tau} must remain strictly between both limit lines $Ideal^- = u \times t - 1$ and $Ideal^+ = u \times t + 1$.

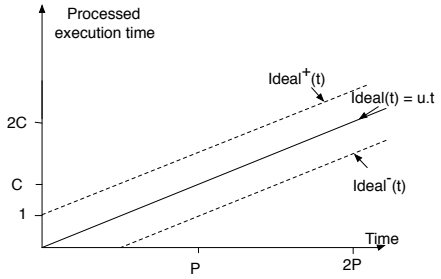

Figure 7: *PFair execution of a task: the execution curve must be located between both dotted lines*

At any time t, a task $\tau = < r, C, D, P >$ is said to be:

- **ahead** if $W_\tau(0,t)$ is above the ideal line $Ideal(t) = u \times t$. It has been processed a little bit more than in the ideal case. We have: $u \times t - \sum_{j=0}^{t-1} S_\tau(j) < 0$.
- **punctual** if it has been processed for exactly $u \times t$ slots. We have: $u \times t - \sum_{j=0}^{t-1} S_\tau(j) = 0$.
- **behind** if $W_\tau(0,t)$ is under the ideal line $Ideal(t) = u \times t$. It has been processed a

little bit less than in the ideal case. We have: $u \times t - \sum_{j=0}^{t-1} S_\tau(j) > 0$.

PFair strategies (they are PFair for each task) follow the global frame described below.

1. The task set is partitioned into three sets.
   - the **Urgent** set collects all the behind tasks which would be too late (under the lower bound) if they were not processed at time t. These tasks must be processed at time t, else the PFairness condition would be violated.
   - the **Tnegru** set collects the ahead tasks which would be too in advance (over the upper bound) if they were processed at time t. These tasks must not be processed at time t, else the PFairness condition would be violated.
   - the **Contending** set collects the other tasks: the PFairness is violated neither if they are processed nor if they are not.
2. Urgent tasks are processed.
3. Contending tasks are sorted.
   The m - |Urgent(t)| first contending tasks are processed[1].

Figure 8 shows examples of PFair and non PFair behaviours, and illustrates the different status of a task.

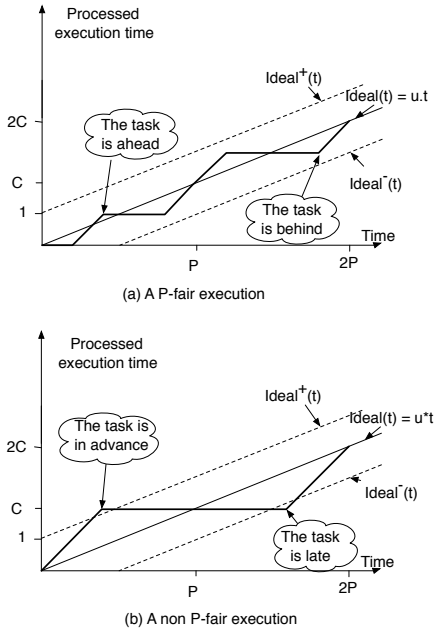

Figure 8: *(a) a PFair execution and (b) a non PFair execution - Different status of the task*

[1] |A| denotes the cardinality of set A.

Several PFair algorithms have been proposed in the litterature (PF, PD and PD^2 (4; 5; 2; 3)). These algorihms differ in the way they select the tasks to process among the contending tasks. These scheduling strategies are very efficient, as stated in theorem 2.

Theorem 2 *(4) The scheduling algorithms PF, PD and PD^2 are optimal for systems of periodic synchronous independent tasks with implicit deadlines (deadlines are equal to periods) in multiprocessor context. Moreover, the system is feasible if and only if $U \leq m$ where m is the number of processors.*

3.2. Extensions of the PFairness

The notion of Pfairness is defined in the context of synchronous tasks with implicit deadlines. In (3; 10), asynchronous systems are considered, where asynchronism means that some of the first task slots do not take place. But the first instance of any task is still assumed to be released at time 0. In (2), sporadic tasks are considered: periods correspond only to the minimum elapsed time between two consecutive releases. We consider here a 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 be useful e.g. 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. We thus consider task systems where tasks may have non zero first release times, and may have constrained deadines ($D \leq P$). To define PFairness in this enlarged context, we first consider the ideal case: in an ideal fair schedule, a task τ must have received at time t, $\omega_\tau(t)$ processor time units (see figure 9) where $\omega_\tau(t)$ is defined by:

$$\omega_\tau(t) = \begin{cases} 0 \; if \; t \in [0, r) & (1) \\ k * C + \frac{C}{D} * (t - k * P - r) \; if \\ t \in [k * P + r, k * P + D + r) & (2) \\ (k+1) * C \; if \\ t \in [k * P + D + r, (k+1) * P + r) & (3) \end{cases}$$

where $k = \lfloor \frac{t}{P} \rfloor$ represents the number of already completed instances of the task.
The condition (1) corresponds to the idleness of the task before its first release, the condition (2) corresponds to the classical PFairness notion, and the part (3) expresses the idleness between each deadline, and the next release. A schedule S is then PFair for a task τ iff

$$\forall t \in \mathbb{N} : -1 < \omega_\tau(t) - \sum_{j=0}^{t-1} S_\tau(j) < 1$$

A schedule is PFair if it is PFair for each task $\tau \in \Gamma$. We can notice that during the idle periods, the task

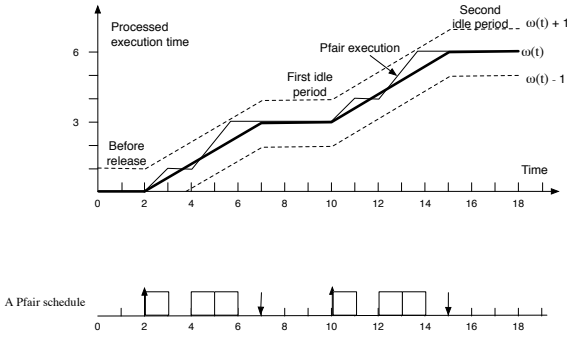

Figure 9: *Ideal progression and a PFair schedule for the task* $< 2, 3, 5, 8 >$

must be punctual, because the only integer points between both limit lines are those located on the ideal line.

3.3. PFair geometric modeling

Our aim is to include PFairness properties in the geometric model. The final goal is to determine the set of the PFair schedules for a given application which cannot be PFairly scheduled by the classical strategies, i.e. when critical resources are used and tasks may have offsets and constrained deadlines. In such context, the PFair algorithms are not optimal. We define the PFair geometric model for a task $\tau = < r, C, D, P >$ as the set of states belonging to a PFair schedule. We call these states the PFair states.

Definition 3 *The PFair geometric model of a task* τ *is the set:*

$$V(\tau) = \{(x,t) \in \mathbb{N}^2 \text{ such that } \exists \text{ a PFair schedule } S_\tau, \; x = \sum_{k=0}^{t-1} S_\tau(k)\}$$

Figure 10 presents the PFair schedule (_ _ _ *p* _ _ _ *p p* _ _ _ _ _) for the task $\tau = < 3, 3, 7, 11 >$ where *p* stands for "the task is processed" and _ for "the task remains idle". In the following, we define this set in a more geometrical way.

3.3.1. Synchronous tasks with implicit deadlines.
We assume that the first release time of a task τ is equal to 0, and that its relative deadline is equal to its period.The discrete model of the PFair behaviours of the task τ is composed of all the states that the task can reach if it is PFairly processed. This set consists of all the discrete points that are close enough (at a distance lower than 1) to the line $x = (C/P)t$. Figure 11 presents the geometrical model $\Omega(\tau)$ of the task $\tau = < 0, 3, 6, 6 >$ and the subset $V(\tau)$ of $\Omega(\tau)$ composed of the PFair states. Let us state

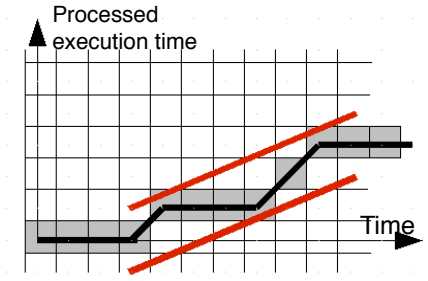

Figure 10: *A PFair schedule of the task* $\tau = < 3, 3, 7, 11 >$

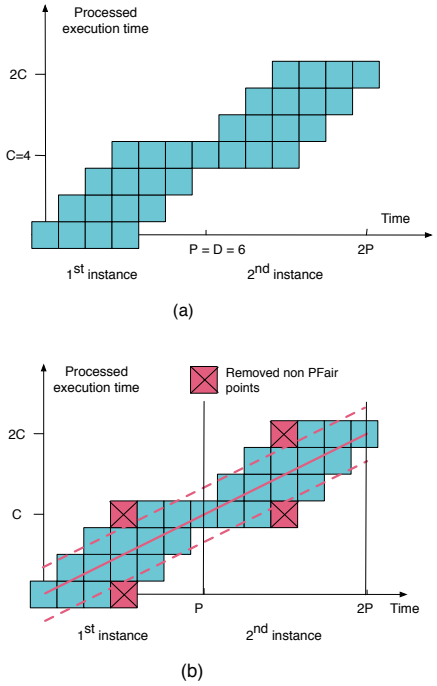

Figure 11: *(a) Geometrical model* $\Omega(\tau)$ *and (b) PFair set* $V(\tau)$ *for the task* $< 0, 3, 6, 6 >$

$x = \sum_{j=0}^{t-1} S_\tau(j)$. From PFairness definition, we have $\forall t \in \mathbb{N}, -1 < \frac{C}{P} \times t - x < 1$, so we get: $\forall t \in \mathbb{N}, -P < C \times t - x \times P < P$. We therefore define the set $V(\tau)$ geometricaly as:

Definition 4 *The set of discrete PFair states for a task* $\tau = < 0, C, D = P, P >$ *is the discrete line of width* 2 *defined by:*

$$V(\tau) = \{(x,t) \in \mathbb{N}^2, -P < Ct - Px < P\}$$

Since PFair states belong to feasible schedules (4), we also have the following property:

$$V(\tau) \subset \Omega(\tau)$$

3.3.2. Asynchronous tasks with implicit deadlines.
We assume that $r > 0$ and $D = P$. In this context, a task cannot be processed before its first release. Therefore, the only states available before this time are those with $x = 0$. After release, the execution curve must remain close to the line $x = \frac{C}{P} \times t + r$. Thus we generalise the previous definition:

Definition 5 *The set of discrete PFair states for a task* $\tau = < r \neq 0, C, D = P, P >$ *is:*

$$V(\tau) = \{(x,t) \in \mathbb{N}^2 s.t.$$

$$\left.\begin{array}{l} \forall t < r, x = 0, \\ \forall t \geq r, C \times r - P < Ct - Px < C \times r + P \end{array}\right\}$$

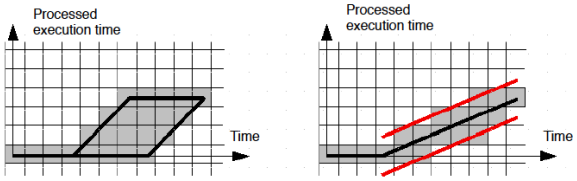

Figure 12: *Geometric model and PFair-state set* $V(\tau)$ *for task* $< 3, 3, 7, 7 >$

The figure 12 presents the geometric PFair model of the task $< 3, 3, 7, 7 >$.

3.3.3. Asynchronous tasks with constrained deadlines.

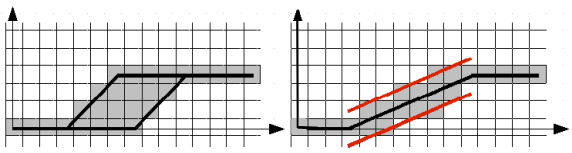

Figure 13: *Geometric model and PFair-state set* $V(\tau)$ *for the task* $< 3, 3, 7, 11 >$

We relax our last constraint and consider a task with a constrained deadline, i.e. such that $D \leq P$. We must model the PFairness condition given in section 3.2. Each instance τ_k of the task must complete its execution before its deadline $d_k = r + (k-1)P + D$ and must be idle between d_k and the release time $r_{k+1} = r + kP$ of the next instance. The task must be processed during C slots between $r_k = r + (k-1)P$ and d_k. So the execution rate is here equal to $CH = \frac{C}{D}$, and the task is idle before r and between d_k and r_{k+1} for each $k \geq 1$. We thus have:

Definition 6 *The set of discrete PFair states for a task* $\tau = < r \neq 0, C, D \leq P, P >$ *is defined by:*

$$V_\tau = \{(x,t) \in \mathbb{N}^2, s.t. \forall k \geq 1$$

$$\left\{\begin{array}{ll} \forall t \leq r, x = 0, & \\ \forall t \in [r_k, d_k[, & \\ Cr_k - D - (k-1)CD & < Ct - Dx \\ & < Cr_k + D - (k-1)CD, \\ \forall t \in [d_k, r_{k+1}[, & x = kC \end{array}\right.$$

Figure 13 illustrates the construction for the task $\tau = < 3, 3, 7, 11 >$.

3.3.4. Asynchronous task systems with constrained deadlines.
We consider now the complete task set $\Gamma = \{\tau_1, \tau_2, \ldots, \tau_n\}$. The PFair geometric model of the application is defined by:

Definition 7 *The set of discrete PFair states for the application* Γ *is*

$$V(\Gamma) = \{(x_1, ...x_n, t) \in \mathbb{N}^{n+1},$$
$$\forall i \in [1, n], (x_i, t) \in V(\tau_i)\}$$

3.3.5. Resource sharing.
We consider a task set $\Gamma = \{\tau_1, \tau_2, , \tau_n\}$. Let R be a critical resource, and $\mathcal{I}_R$ be such that τ_i uses R iff $i \in \mathcal{I}_R$. We must eliminate from the model all the invalide states. These states correspond to:

1. The joint use by two tasks of a critical resource.
2. Isolated states, which have no predecessor. They thus don't belong to a PFair schedule.
3. Dumb states, which have no successor in the model. They neither belong to a PFair schedule.

We first consider point 1. We remove all the points corresponding to a violation of the mutual exclusion rules. For a given resource R, these R-invalide points are collected in a set $\overline{\Omega(R)}$. Then, we remove iteratively isolated and dumb points, until a fixed point is reached. For simplicity reasons, we assume that each task using a resource R contains only one critical section for R. The definitions can then be easily extended to several critical sections. If a task τ uses a resource R, we denote $pre_\tau(R)$ (resp. $post_\tau(R)$) the processing time until the begining (resp. the end) of the critical section. The critical section lasts thus $post_\tau(R) - pre_\tau(R)$. We define then the set of the R-invalide points by:

Definition 8 *The set of the R-invalide states is defined as:*

$$\overline{\Omega(R)} = \{(x_1, \ldots, x_n, t) \in \Omega(\Gamma)\ s.t.$$

$$\begin{cases} \exists j_1, j_2 \in \mathcal{I}_R, \\ \lfloor \frac{x_{j_1}}{P_{j_1}} \rfloor . P_{j_1} + pre_{\tau_{j_1}}(R) & < x_{j_1} \\ & < \lfloor \frac{x_{j_1}}{P_{j_1}} \rfloor . P_{j_1} + post_{\tau_{j_1}}(R) \\ and \\ \lfloor \frac{x_{j_2}}{P_{j_2}} \rfloor . P_{j_2} + pre_{\tau_{j_2}}(R) & < x_{j_2} \\ & < \lfloor \frac{x_{j_2}}{P_{j_2}} \rfloor . P_{j_2} + post_{\tau_{j_2}}(R) \end{cases}$$

The set of the discrete resource-valid points is the set:

$$\Omega(\Gamma, \mathcal{R}es) = \Omega(\Gamma) \setminus \cup_{R \in \mathcal{R}es} \overline{\Omega(R)}$$

And the set of the discrete PFair resource-valid points is the set:

$$Val(\Gamma) = V(\Gamma) \setminus \cup_{R \in \mathcal{R}es} \overline{\Omega(R)}$$

where $\mathcal{R}es$ is the set of all the critical resources used by the application.

The inequalities express the fact that both pending instances of the tasks τ_{j_1} and τ_{j_2} are in their critical section. $\lfloor \frac{x_{j_1}}{P_{j_1}} \rfloor$ (resp. $\lfloor \frac{x_{j_1}}{P_{j_1}} \rfloor$) is the number of already completed instances of τ_{j_1} (resp. τ_{j_2}). We then remove isolated and dumb points: a point $(x_1, \ldots, x_n, t)$ is isolated if there in no point $(x'_1, \ldots, x'_n, t-1)$ with $x'_i = x_i$ or $x'_i = x_i - 1$ and a point $(x_1, \ldots, x_n, t)$ is dumb if there is no point $(x'_1, \ldots, x'_n, t+1)$ with $x'_i = x_i$ or $x'_i = x_i + 1$. We can note that an isolated point is the n-neighbour of no point, and a dumb point has no n-neighbour.
If we restrict ourselves to the time interval $[0, max([r_i]_{i\ 1 \ldots n}) + 2 * H]$, the number of points is finite, thus, we are insured to get a fixed-point when we iterate the removing process. We thus finaly get a model denoted by $M(\Gamma, T)$ if we consider the time interval $[0, T]$.

3.3.6. Elements of complexity

The number of states is in the worst case equal to $nb_i = r_i + (C_i.D_i + P_i - D_i) \times \frac{T - r_i}{P_i}$ for one task, and for the complete model, it is at most equal to $\prod_{i=1}^{n} nb_i$. Then each further construction step (construction of $\overline{\Omega(R)}$ and of $Val(\Gamma)$) is linear in the number of states. And finally, since, once the model constructed, each analysis step is linear in T.

4. SYSTEM ANALYSIS USING THE GEOMETRIC MODEL

In the previous section, we have proposed a model $M(\Gamma, T)$ which can take temporal validity, mutual exclusion properties and PFairness into account. We present now some examples of the use of the model for the sake of temporal and qualitative analysis of a real-time application. The general frame of our analysis consists in the use of the model to compute the set of feasible schedules with additional PFairness properties.

4.1. Global PFairness

Firstly, we can use the model in order to generate all the PFair behaviours. We recall that in our context: asynchronous systems with constrained deadlines and critical resources, there exists no on-line optimal strategy. We use the geometric model to determine whether there exists some PFair schedules for the application, for a given architecture. We finally have the following result:

Proposition 9 *There exists a PFair schedule for a platform of m identical processors on the time interval $[0, T]$ iff there exists a m-connected path (of length$T$) in the model $M(\Gamma, T)$.*

4.2. Partial Pfairness

Considering global PFairness may be too restricting. First, it can be useful to insure processing regularity for some tasks, but not for all. And requiring PFairness for all the tasks may lead to unfeasibility, due e.g. to the constaints induced by the resource management. When PFair strategies (PF, PD or PD^2) are considered, PFairness requirements are globally applied. Our methode enables us to refine the analysis, and to restrict the PFairness requirements to a subset of tasks. This leads to more flexible scheduling. A direct application is the possibility to combine an off-line method to schedule the periodic tasks with an efficient on-line algorithm for the integration of aperiodic tasks with strict deadlines. We have proposed in (7) an on-line acceptation algorithm which relies on a PFair distribution of processor idle times. The presented methodology uses a PFair scheduling algorithm to schedule the periodic application. We make the assumption that the tasks are synchronous, independent, with implicit deadlines, and that $m - 1 < U < m$. In such a case, the processors remain idle for $H(m - U)$ processor time units. A further task, the idle task is added to the system. It is defined by $\tau_0 =< 0, H(m-U), H, H >$. The extended application is then globally scheduled by means of a PFair strategy. It follows that a PFair distribution of the processor idle times is guarantied. Here, we can enlarged the acceptation routine to applications composed of non independent periodic tasks which may be asynchronous and have constrained deadlines. We then add the idle task to the system, and we require the idle task, and only it, to be PFairly processed. The acceptation routine for aperiodic trafic can then be used even if the other periodic tasks are not PFairly scheduled.

4.3. Non PFairly feasible systems

For some applications, because of the use of critical resources, it is definitively not possible to find PFair

schedules. This means that $M(\Gamma, T)$ contains no m-connected paths. In this case, it can nevertheless be interesting, for quality reasons, to find a schedule which is as fair as possible, provided $\Omega(\Gamma, \mathcal{R}es)$ contains m-connected paths. Here again, this requirement can be reduced to a subset of tasks. For that purpose, we use a PFairness benchmark. In (8), we have proposed several PFairness measures for synchronous task systems, with implicit deadlines. We enlarge the most promising one to our context. For synchronous tasks with implicit deadlines, the on-line PFair strategies (PF, PD and PD[2]) rely on the decomposition of the execution of each task $\tau =< r, C, D, P >$ into the execution of unitary subtasks τ^j ($j > 0$). Each subtasks τ^j has a pseudo-release time and a pseudo-deadline defined by: $r^j = \lfloor \frac{j-1}{u(\tau)} \rfloor$ et $d^j = \lceil \frac{j}{u(\tau)} \rceil$. The time interval $[r^j, d^j)$ is the feasability window of the substask. The task τ is PFairly processed iff each of its subtask is scheduled within its feasability window. The PFairness measure that we propose depends on the distance of the execution time of the subtasks to their feasability windows. This distance is equal to 0 if the execution is PFair. We must enlarge the definition of the measure to asynchronous task sets with constrained deadlines. We first adapt the notion of feasability window. In our context, the feasability window of the subtask τ^j is defined by: $[r^j, d^j)$ with $r^j = r + k.P + \left\lfloor \frac{j-1-k.C}{CH(\tau)} \right\rfloor$ and $d^j = r + k.P + \left\lceil \frac{j-k.C}{CH(\tau)} \right\rceil$ where $CH(\tau) = \frac{C}{D}$ is the density of the tasks[2] and $k = 1 + \lfloor \frac{j-1}{C} \rfloor$ is the instance number of the task τ. We define then for a schedule S the distance between a subtask and its feasibility window by: $dist(\tau^j, S) = Max\{0, r^j - b^j, e^j - d^j\}$ where b^j is the begining date of the subtask τ^j in the schedule S and e^j is its terminaison date ($e^j = b^j + 1$). We can see in figure 14 an execution of the two first instances of a task $\tau =< 2, 3, 5, 8 >$, thus the 6^{th} first subtasks. We can see within each subtask its distance to its feasability window, which is equal to 0 for the subtasks τ^1, τ^2, τ^4 and τ^5, and equal to 1 for the subtasks τ^3 (it is processed one time unit too early) and τ^6 (it is processed one time unit too late). The PFairness measure for a given schedule, a given task and a time interval $[0, T]$ is the quotient of the cumulated distances of each subtask whose feasability interval in included within $[0, T)$, and of the worst possible value. The worst value is obtained when each instance τ_k is processed either at the beginning or at the end of its execution window

[2] If the task has an implicit deadline, its density equals its utilisation factor

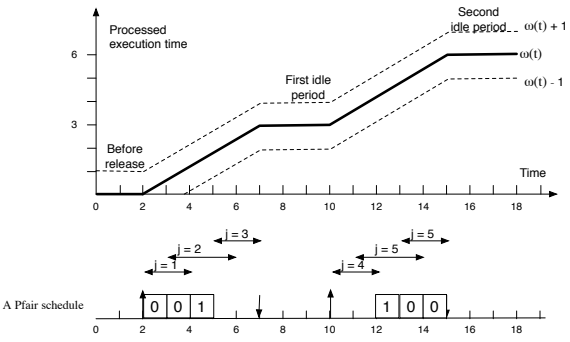

Figure 14: *Distance beetwen a subtask and its feasability window*

$[r + (k-1)P, r + (k-1)P + D]$. Formally, we have:

$$d_{max} = nbr(\tau, T) \times \sum_{i=1}^{C} (\lceil \frac{i}{CH(\tau)} \rceil - (i-1)) + \sum_{i=1}^{n(\tau,T) mod C} (\lceil \frac{i}{CH(\tau)} \rceil - (i-1))$$

and

$$Measure(S, \tau, T) = \frac{\sum_{j=1}^{n(\tau,T)} dist(\tau^j, S)}{d_{max}}$$

where

- $n(\tau, T)$ is the number of subtasks whose feasability windows are in $[0, T]$
- $nbr(\tau, T)$ is the number of complete periods within $[0, T]$. We have $nbr(\tau, T) = \lfloor \frac{T-r}{P} \rfloor$

5. CONCLUSION

PFair scheduling is a very powerful strategy for real-time scheduling in multiprocessor context. It has been proved to be optimal for synchronous, independent task systems, with implicit deadlines. It could thus be of interest to use it for larger context. Furthermore, it can be interesting to enforce PFair behaviours for some kind of applications. PFair constraints can also be restricted to a subset of tasks. Besides, the discrete geometrical approach is really effective to decide feasability and finding feasible schedules. We have here coupled both notions. We have presented the geometrical characterisation of PFair behaviours of a real-time application. We have then explained how to take resource management constraints into account in our model. Finally, we have presented three applications of our modeling methodology for the analysis of a real-time application with PFairness requirements.

6. REFERENCES

[1]B. Andersson and J. Jonsson. Fixed-priority preemptive multiprocessor scheduling: to partition or not to partition. In *Proceedings of the conference on Real-Time Computing Systems and Applications*, pages 337–346, December 2000.

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

[3]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.

[4]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.

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

[6]G.C. Buttazzo. *Hard Real-Time Computing Systems*. Kluwer Academic Publishers

[7]A. Choquet-Geniet, C. Fotsing, and S. Malo. Scheduling of real-time applications with variable utilization factor using a pfair based aperiodic server. Technical report, LISI, ENSMA and University of Poitiers, March 2009.

[8]A. Choquet-Geniet, G. Largeteau-Skapin, and A. Ouattara. Mesure de l'équité d'une application temps à l'aide d'une approche à base de géométrie discrète. *JESA*, 43:1065–1080, 2009.

[9]L. Cucu and J. Goossens. Feasibility intervals for fixed-priority real-time scheduling on uniform multiprocessors. In *11th IEEE international Conference on Emerging Technologies and Factory Automation*, pages 397–405. IEEE Computer Society Press Ed., 2006.

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

[11]Geniet D. and Largeteau-Skapin G. Wcet free time analysis of hard real-time systems on multi-processors: a regular language-based model. *Thoretical Computer Science*, 388:26–52, 2007.

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

[13]E. Grolleau and A. Choquet-Geniet. Scheduling real-time systems by means of Petri nets. In *Proc. of* 25^th *Workshop on Real-Time Programming*, pages 95–100. Universidad Politécnica de Valencia, 2000.

[14]K. Hong and J. Leung. On-line scheduling of real-time tasks. *IEEE transactions on Computers*, 41(10):1326–1331, 1992.

[15]G. Largeteau, D. Geniet, and E. Andres. Discrete geometry applied in hard real-time systems validation. In *Proc. of* 12^{th} *Discrete Geometry for Computer Imagery*, LNCS 3429, pages 23–33. Springer Verlag, 2005.

[16]G. Largeteau, D. Geniet, and J.P. Dubernard. Validation of distributed periodic real-time systems using can protocol with finite automata. In *Proc. of* 5^{th} *S.C.I.* I.I.I.S., 2001.

[17]J. Leung and J. Whitehead. On the complexity of fixed-priority scheduling of periodic real-time tasks. *Performance Evaluation*, pages 237–250, 1982.

[18]S. Malo and A. Choquet-Geniet. Analysis of critical scalable real-time systems by means of petri nets. In L. O'Reilly and M. Roggenbach, editors, *Proceedings of the Ninth International Workshop on Automated Verification of Critical Systems*, pages PP. 245–247, 2009.