A New Method for Determining Fixed Priority Configurations for Real-Time Systems on Multiprocessor Targets

Bernard Chauvière, LISI, ENSMA, F-86961 Futuroscope Chasseneuil Cedex
Dominique Geniet, LISI, Univ. Poitiers, F-86961 Futuroscope Chasseneuil Cedex
René Schott, IECN and LORIA, Univ. H.Poincaré, F-54506 Vandœuvre-les-Nancy Cedex

***Abstract*— Several methods have been developed for determining fixed priority configurations on uniprocessor. In this paper, we present a new method, called PFX, for determining fixed priority configurations on multiprocessor. PFX follows an off-line approach: our constructive process uses an inductive analysis of the whole set of possibilities. Therefore, PFX provides all valid fixed priority configurations. Experimentations (section V) show that PFX is very efficient on multiprocessor and subsumes Rate Monotonic (RM), Deadline Monotonic (DM) as well as Earliest Deadline First (EDF). In addition, we have found task systems which can be scheduled by PFX, but can not be scheduled by RM, DM, EDF or Least Laxity First (LLF).**

***Index Terms*— Real-Time, Fixed priority scheduling, Multiprocessor, Asynchronous periodic tasks, Hard deadlines**

I. Introduction

The scheduling of real-time systems relies upon the use of operating systems (for example VxWorks). Usually, these operating systems use fixed priority scheduling: the priorities are attributed to the tasks at the start and remain fixed. Priorities are generally attributed as follows: either the designer decides the priorities according to the needs of the system and to his experience, or he uses a fixed priority policy which determines the priority of each task.

Many policies have been proposed for uniprocessor scheduling: for example Rate Monotonic (RM) and Earliest Deadline First (EDF) [15], Deadline Monotonic (DM) [14] and Least Laxity First (LLF) [12]. EDF and LLF are optimal on uniprocessor, DM is optimal on uniprocessor when the tasks start simultaneously and RM is optimal on uniprocessor for synchronous task systems with implicit deadline. Necessary and sufficient conditions have been proved for a policy to produce reliable scheduling of a task system for RM [13] and for EDF [15]. The reader interested in real-time scheduling surveys can read [4][16].

[12] shows that no scheduling policy is optimal on multiprocessor. In this context, we observe the following qualitative results. RM and DM are low level performing and then must be used with care. Observing our experimentation results show that EDF is also a low level performance policy. By the way, LLF is not an event-driven policy and its implementation is therefore not realistic. [6] proposes a new algorithm called PFair whose optimality has been proved for synchronous task systems with implicit deadline. PFair has the same drawback as LLF.

Another strategy to solve the scheduling problem for a task system consists in finding fixed priority configurations which make it feasible. [3] proposes such an approach for uniprocessor. This method is optimal, and it is useful to find fixed priority configurations different from RM or DM priorities. [1] proves that this method is not optimal for the multiprocessor context, but [7] uses it to produce multiprocessor fixed priority configurations which are different from RM priorities.

In this paper we present an off-line method we call PFX which generates recursively (with respect to the length of execution sequences) all valid fixed priority configurations. We limit the complexity level of the process by using a basis concept which merges all priority configurations which generate sequences with identical prefixes. The time complexity of PFX depends on the number of solutions, not on the size of the task system. Our experimentations confirm this result, since the computation time decreases when the load of the task systems increases.

Our method produces all valid fixed priority configurations. It is therefore optimal for attributing fixed priorities in the multiprocessor context. The knowledge of all valid configurations makes the designer able to select the best one according to specific criteria: response time, jitter, etc. Following the same technique that [7], we have found PFX-schedulable task systems which can not be scheduled using RM or DM (and also, sometimes, using EDF or LLF): we are able to produce a valid fixed priority configuration for these systems using our technique. PFX deals with the following context: multiprocessor, asynchronous systems, hard deadline. It produces solutions which are not reached using others scheduling techniques.

This paper is organized as follows: Section II presents the software and hardware context we deal with. Then, Section III gives the principles of our approach, and Section IV provides practical implementation aspects. The experimentation section concludes the paper: we compare the scheduling power of our method with classical approaches (RM, DM, EDF, LLF) and with the methods proposed in [5] and [2]. Results are surprising!

II. Context

A. Software context

We consider real-time systems composed of a (static) finite number of independent periodic tasks. The current task system is denoted by Ξ in the following.

Each task $\tau_i \in \Xi$ is a program designed to react to a specific event e_i. Each occurrence $e_{i,j}$ of e_i is associated with its occuring-time $t_{i,j}$. The software reaction to $e_{i,j}$ consists in creating a job $\tau_{i,j}$ based on task τ_i – i.e. running τ_i's code – that is activated at time $t_{i,j}$. If we get $\exists T_i \in \mathbb{N}^*$ such that $\forall j \in \mathbb{N}, t_{i,j} = r_i + jT_i$, the task τ_i is called *periodic*, and T_i is called *period* of τ_i. If not, τ_i is called *aperiodic*. In this paper, we deal with periodic tasks.

Each task τ_i is characterized by the following time attributes:

- r_i is the first release time, i.e. the time $t_{i,0}$ associated with the first occurrence of e_i,
- C_i is the worst case execution time (WCET),
- D_i is the relative deadline, i.e. the time delay between the release time and the deadline of each job of τ_i,
- T_i is the period.

We consider that all temporal parameters are natural numbers. All jobs of τ_i share the attributes C_i and D_i. The release time of the job $\tau_{i,j}$ is $r_{i,j} = r_i + jT_i$, and its absolute deadline is $d_{i,j} = r_{i,j} + D_i$. A task can not be parallelized, so $C_i \leq D_i$. In addition, we assume that tasks are not reentrent: $D_i \leq T_i$.

B. Hardware context

Real-time applications can be hosted on various hardware systems. Three different contexts are typical: uniprocessor, multiprocessor and distributed architectures. In the distributed context, processors are not directly interconnected: communications stand on a network. So delays induced by networking have to be considered as scheduling constraints. Such parameters used to be not deterministic, because they depend on the networking behaviours. On the opposite, the multiprocessor context corresponds to the use of memory sharing (PRAM model): process communication is completed in $O(1)$ time.

Two different approaches (partitioned and global) have been developed for the study of multiprocessor architectures. In the partitioned approach, tasks are attributed beforehand to each processor. So the multiple instance problem is equivalent to a uniprocessor scheduling problem. Nevertheless, the static attribution of tasks on processors is NP-hard [8]. Conversely, the global approach allows the different jobs of the same task to be executed on different processors. Under the total migration assumption, dynamic processor changes are possible for the same job. [8] gives a classification of the different multiprocessor scheduling problems.

In this paper we deal with multiprocessor architectures via the global approach. We assume that the global migration assumption is fulfilled: jobs can migrate at any time. In addition, tasks are supposed to be preemptive: a task can be suspended during execution in order to allow the treatment of a higher priority one. In the following, p is the number of processors of the currently used target.

C. Scheduling Sequences

A scheduling sequence models the execution of a task system. At every time, a scheduling sequence collects the set of running tasks, i.e. tasks which own a processor. This section presents some definitions concerning this concept.

Time is modeled by the set $\mathbb{R}^+$: 0 corresponds to the init of the real-time software we are concerned with. We assume that if no event occurs on a time interval $[t_1, t_2[$, then there are no changes in the set of running tasks between t_1 and t_2. We also assume that there exists a delay $u \in \mathbb{R}^+$ such that if two consecutive events e_1 and e_2 occur at times t_1 and t_2, then $t_2 - t_1 \in u\mathbb{N}$. u is the minimal observation capacity of our system. For instance, it may be the duration of any atomic statement on the considered target architecture. We consider that u is our time unit, i.e. that $u = 1$. Then, observing the evolution of the system at time $t \in \mathbb{N}$ is equivalent to observing it at any time in $[t, t+1[$.

This is why we consider discrete time: the integer t corresponds to the time interval $[t, t+1[$. For simplicity, we denote by $\mathbb{N}_t$ the set $[0, t] \cap \mathbb{N}$, and by $\mathbb{N}_t^\star$ the set $[1, t] \cap \mathbb{N}$.

Definition 1: A scheduling sequence $(s_t)_{t \in \mathbb{N}}$ is a sequence of subsets of Ξ. For every time $t \in \mathbb{N}$, the set s_t collects all tasks running at time t.

We denote by S_Ξ the set of scheduling sequences of Ξ.

Example 1: Consider the following task system, called cnf_one:

Task	r_i	C_i	D_i	T_i
τ_1	0	1	3	3
τ_2	0	2	6	6
τ_3	0	1	4	4

One possible 2-processor scheduling sequence for this example is the permanent iteration of the sequence presented in Figure 1.

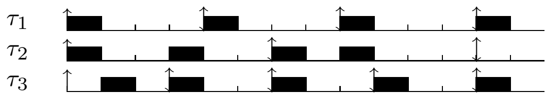


Fig. 1. Scheduling sequence for cnf_one

A task $\tau_i \in \Xi$ is active if its current job is not yet completed, so some statements are pending. Active tasks can be defined thanks to the total CPU-owning time of their successive jobs.

Definition 2: Let $s \in S_\Xi$ be a scheduling sequence and $t \in \mathbb{N}$. For any job $\tau_{i,j}$, we get:

$$C_{i,j}(s,t) = \sum_{u=r_{i,j}}^{u=t-1} |s_u \cap \{\tau_i\}|$$

Remark that for $t \leq r_{i,j}$, we get $C_{i,j}(s,t) = 0$.

We can now define the notion of active task.

Definition 3: Let $s \in S_\Xi$ be a scheduling sequence and $t \in \mathbb{N}$. The task $\tau_i \in \Xi$ is active at time $t \in \mathbb{N}$ if and only if:

$$t \geq r_i \text{ and } C_{i,\left\lfloor \frac{t-r_i}{T_i} \right\rfloor}(s,t) < C_i$$

We denote by s_t^a the set of active tasks at time t in the scheduling sequence s.

Example 2: Gantt's chart presented in Figure 1 gives the tasks running at every time. Table I completes this chart with the set of active tasks and the value of the total CPU-owning time acquired by each task.

t	s_t^a	s_t	$C_{i,\left\lfloor \frac{t-r_i}{T_i} \right\rfloor}(s,t)$		
			τ_1 $(i=1)$	τ_2 $(i=2)$	τ_3 $(i=3)$
0	$\{\tau_1,\tau_2,\tau_3\}$	$\{\tau_2,\tau_3\}$	0	0	0
1	$\{\tau_1,\tau_2\}$	$\{\tau_1\}$	0	1	1
2	$\{\tau_2\}$	$\emptyset$	1	1	1
3	$\{\tau_1,\tau_2\}$	$\{\tau_1,\tau_2\}$	0	1	1
4	$\{\tau_3\}$	$\{\tau_3\}$	1	2	0
5	$\emptyset$	$\emptyset$	1	2	1
6	$\{\tau_1,\tau_2\}$	$\{\tau_1,\tau_2\}$	0	0	1
7	$\{\tau_2\}$	$\emptyset$	1	1	1
8	$\{\tau_2,\tau_3\}$	$\{\tau_2,\tau_3\}$	1	1	0
9	$\{\tau_1\}$	$\{\tau_1\}$	0	2	1
10	$\emptyset$	$\emptyset$	1	2	1
11	$\emptyset$	$\emptyset$	1	2	1

TABLE I
CHARACTERISTICS OF THE SCHEDULING SEQUENCE s DEFINED IN FIGURE 1

A specific scheduling is planned to be executed on a specific architecture. Its feasibility involves its ability to satisfy some constraints depending on the architecture. In this work, we deal with PRAM architectures, so the only characteristic we must deal with is the limitation of the number of processors. Remember that p denotes the number of available processors.

A scheduling sequence is *compatible* with a p processor architecture if it never plans to execute more than p tasks simultaneously. Moreover, a scheduling generated for a real-time operating system is work-conserving: no processor can be idle when there are pending tasks. In the following, we are interested in this kind of scheduling. The next definition formalizes all these notions.

Definition 4: Let $s \in S_\Xi$.

- s is p-compatible if and only if $\forall t \in \mathbb{N}, |s_t| \leq p$.
- s is p-conservative if and only if $\forall t \in \mathbb{N}, |s_t| = min\left\{p, |s_t^a|\right\}$.
- Let $t \in \mathbb{N}$, s is (p,t)-conservative if and only if:

$$\forall u < t, |s_u| = min\left\{p, \left|s_u^a\right|\right\}$$

We denote by S_Ξ^p the set of p-compatible scheduling sequences of Ξ.

D. Fixed Priority Configurations

A scheduling algorithm keeps the set of active tasks sorted following a specific criterion, called *priority*. Priorities are implemented as numeric values, then sorting tasks stands on the natural order on $\mathbb{R}$.

Priorities can be *static* or *dynamic*. When they are static, the value associated with each task τ_i is defined before starting the execution, and can not be modified while the software runs. Then all jobs $\tau_{i,j}$ concerning task τ_i are associated with the same priority. Moreover, τ_i and τ_j are associated with different values. This strategy is called *fixed priority configurations*. So a fixed priority configuration corresponds to a total order on the task set.

RM and DM are called *fixed priority policies*. However, they do not satisfy all properties of fixed priority configurations: RM and DM can assign the same priority to two different tasks (e.g. when periods are equal), therefore they do not involve a total order. This is why a second criterion is usually associated with these policies, to make the scheduling process deterministic: for instance numbers of tasks can be used. Hence, a fixed priority policy can be associated with an equivalent fixed priority configuration, but it is not true in general.

This is why one must distinguish fixed priority *policies* from fixed priority *configurations*: configurations reach all fixed priority schedulings, but policies may not. In this work, we focus on computing configurations, and we do not deal with looking for policies corresponding to specific configurations. In that follows, we present the basic notions on fixed priority configurations.

Definition 5: A fixed priority configuration on Ξ is a bijection $A : \Xi \to \mathbb{N}^\star_{|\Xi|}$.

We denote by $\mathcal{C}_\Xi$ the set of fixed priority configurations on Ξ.

Example 3: For the task system cnf_one, the RM policy associates $\frac{1}{3}$ with τ_1, $\frac{1}{6}$ with τ_2 and $\frac{1}{4}$ with τ_3. This is equivalent to $\tau_1 \leftrightarrow 3$, $\tau_2 \leftrightarrow 1$ and $\tau_3 \leftrightarrow 2$. For this task system, we can use RM or DM priorities, or any other equivalent priority configuration:

RM			DM			Other…		
τ_1	$\leftrightarrow$	3	τ_1	$\leftrightarrow$	3	τ_1	$\leftrightarrow$	3
τ_2	$\leftrightarrow$	1	τ_2	$\leftrightarrow$	1	τ_2	$\leftrightarrow$	2
τ_3	$\leftrightarrow$	2	τ_3	$\leftrightarrow$	2	τ_3	$\leftrightarrow$	1

Each fixed priority configuration leads the scheduling algorithm to produce a specific scheduling sequence, but these sequences are identical as soon as the priorities assigned to two different tasks are never compared by the scheduler (for example, when two tasks can never be active simultaneously).

To produce the sequence corresponding to a specific configuration, the scheduling algorithm chooses at each time the p higher priority tasks among the set of active tasks. The following definition shows how scheduling sequences are associated with fixed priority configurations.

Definition 6: Let $A \in \mathcal{C}_\Xi$ be a fixed priority configuration on Ξ. The scheduling sequence on Ξ generated by A on a p processor architecture is defined as follows:

$\forall t \in \mathbb{N}$,

$$s\left(A,p\right)_t = \left\{\tau_i \in s\left(A,p\right)_t^a \text{ such that } \left|E_i \cap s\left(A,p\right)_t^a\right| < p\right\}$$

where E_i is the set

$$E_i = \left\{\tau_j \in \Xi \text{ such that } A\left(\tau_j\right) > A\left(\tau_i\right)\right\}$$

Note that a sequence $s\left(A,p\right)$ generated thanks to a fixed priority configuration $A \in \mathcal{C}_\Xi$ is always p-conservative.

III. Fixed Priority Scheduling

Scheduling tasks using classical policies (RM, DM, etc.) is useful, because these policies are implemented into real-time operating systems. They are optimal for the uniprocessor context. Unfortunately, the optimality results are no longer valid in the multiprocessor context, and these policies are significantly less efficient. Searching alternative fixed priority configurations may lead to limit this loss of performance, and can produce solutions compatible with existing operating systems.

In this section, we characterize the sequences which can be generated thanks to a fixed priority scheduling algorithm, and we show how the corresponding fixed priority configuration can be produced.

A. Priority Relations

A fixed priority configuration which generates a scheduling sequence s can be computed if and only if a configuration $A \in \mathcal{C}_\Xi$ which satisfies $s = s\left(A,p\right)$ exists. Searching A leads to search a binary transitive relation on Ξ. In this section, we characterise the binary relation according to a specific scheduling.

Choosing τ_i when both τ_j and τ_k are also active involves that the priority level of τ_i is maximal in the set $\left\{\tau_i, \tau_j, \tau_k\right\}$. Observing these properties while following a sequence s can lead to a fixed priority configuration compatible with s. This computing process stands on the notion of priority relation.

Definition 7: Let $A \subset \Xi$ be a set of active tasks and $B \subset A$ its subset of running tasks. We call priority relation induced by the sets A and B every couple which belongs to the set $RP_A\left(B\right)$ defined in the following way:

$$RP_A\left(B\right) = \left\{\left(\tau_i, \tau_j\right) | \tau_i \in B \text{ and } \tau_j \in A \setminus B\right\}$$

The following definition gives the induced priority relations at time t for a given execution sequence.

Definition 8: Let $s \in S_\Xi$ and $t \in \mathbb{N}$. We call (s,t)-induced priority relations on Ξ the set defined as follows:

$$RP_t\left(s\right) = RP_{s_t^a}\left(s_t\right)$$

Example 4: For the scheduling sequence of the task system cnf_one (see Example 1), we get the following (s,t)-induced priority relation sequence on the time interval $\mathbb{N}_{11}$:

t	$RP_t(s)$	t	$RP_t(s)$
0	$\{(\tau_2,\tau_1),(\tau_3,\tau_1)\}$	6	$\emptyset$
1	$\{(\tau_1,\tau_2)\}$	7	$\emptyset$
2	$\emptyset$	8	$\emptyset$
3	$\emptyset$	9	$\emptyset$
4	$\emptyset$	10	$\emptyset$
5	$\emptyset$	11	$\emptyset$

As far as we consider fixed priority context, belonging to a specific $RP_t(s)$ leads a condition to stand for every $u > t$, i.e. for the whole life of the software. This is why we define the priority relations induced by a sequence as follows (recall that the transitive closure of a binary relation R is denoted by $\overline{R}$).

Definition 9: Let $s \in S_\Xi$. We call s-induced priority relations on Ξ the set:

$$RP(s) = \overline{\bigcup_{t\in\mathbb{N}} RP_t(s)}$$

Example 5: For the scheduling sequence s presented in Figure 1, we get the following s-induced priority relations (see the table of $RP_t(s)$'s on Example 4):

$$RP(s) = \{(\tau_1,\tau_2),(\tau_2,\tau_1),(\tau_3,\tau_1),(\tau_3,\tau_2)\}$$

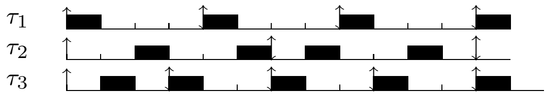


Fig. 2. Scheduling sequence for cnf_one

Let us now consider the uniprocessor scheduling sequence s' concerning cnf_one and presented in Figure 2. The s-induced priority relation computing leads us to

$$RP(s') = \{(\tau_3,\tau_1),(\tau_3,\tau_2),(\tau_1,\tau_2)\}$$

Note that s and s' induce different priority relations. s' can be generated thanks to fixed priorities as soon as task priorities satisfy $RP(s')$. The configuration $\tau_1 \leftrightarrow 2$, $\tau_2 \leftrightarrow 1$ and $\tau_3 \leftrightarrow 3$ is a possible solution. On the opposite, s can not be generated in this way, because a valid fixed priority configuration should satisfy all relations in $RP(s)$. This is not possible, because all tasks should share the same priority level, that is incompatible with fixed priority scheduling.

Therefore, no fixed priority scheduling can generate a scheduling sequence s as soon as $RP(s)$ contains a cycle.

B. Fixed Priority Relations

In this section, we present a necessary and sufficient condition for a sequence s to be produced from a fixed priority policy. We first extend the notation RP to fixed priority configurations.

Definition 10: Let $A \in \mathcal{C}_\Xi$. We call set of priority relations induced by A the set

$$RP(A) = \left\{(\tau_i,\tau_j) \in \Xi^2 \text{ such that } A(\tau_i) > A(\tau_j)\right\}$$

By construction, $RP(A)$ does not contain cycles.

Lemma 1: Let $A \in \mathcal{C}_\Xi$. We get $RP(s(A,p)) \subset RP(A)$.

Proof

Let $t \in \mathbb{N}$ and $(\tau_i,\tau_j) \in RP_t(s(A,p))$. The definition of $RP_t(s)$ leads to $\tau_i \in s(A,p)_t$ and $\tau_j \in s(A,p)^a_t \setminus s(A,p)_t$. The definition of $s(A,p)$ leads to $A(\tau_i) > A(\tau_j)$, and then to $(\tau_i,\tau_j) \in RP(A)$.

QED

Theorem 1: Let $s \in S_\Xi$, and $A \in \mathcal{C}_\Xi$ such that $s(A,p) = s$. Then s is p-conservative and $RP(s)$ does not contain cycles.

Proof

Let A be a fixed priority configuration such that $s(A,p) = s$. Using Lemma 1, we get $RP(s) = RP(s(A,p)) \subset RP(A)$. Since $RP(A)$ does not contain cycles, this property stands also for $RP(s)$. Moreover, $s(A,p)$ is p-conservative, hence the property also stands for s.

QED

Lemma 2: Let $A \in \mathcal{C}_\Xi$ and $s \in S_\Xi$ be a p-conservative sequence such that $RP(s) \subset RP(A)$. For $t \in \mathbb{N}$, we get:

$$s^a_t = s(A,p)^a_t \Rightarrow s_t = s(A,p)_t$$

Proof

Let $t \in \mathbb{N}$ such that $s^a_t = s(A,p)^a_t$. Since s is a p-conservative sequence, $s(A,p)_t = s(A,p)^a_t$ leads to $s_t = s^a_t$, and then to $s_t = s(A,p)_t$.

Let us now suppose that $s(A,p)_t \neq s(A,p)^a_t$. Then there are two tasks τ_i and τ_j such that

$$\begin{cases} \tau_i \in s(A,p)_t \\ \tau_j \in s(A,p)^a_t \setminus s(A,p)_t \end{cases}$$

By definition of $s(A,p)$, every couple (τ_i,τ_j) which satisfies this property satisfies also $A(\tau_i) > A(\tau_j)$. By definition of $RP(A)$, we get $(\tau_i,\tau_j) \in RP(A)$. We know that $RP(A)$ does not contain cycles, and the lemma assumes $RP(s) \subset RP(A)$. Then (τ_j,τ_i) can not belong to $RP(s)$. Consequently, $\tau_j \notin s_t$ or $\tau_i \notin s^a_t \setminus s_t$. Thus we get the following possible cases:

1) $\{\tau_i,\tau_j\} \subset s_t$
2) $\{\tau_i,\tau_j\} \subset s^a_t \setminus s_t$
3) $\tau_i \in s_t$ and $\tau_j \in s^a_t \setminus s_t$

Case 1

Since s and $s(A,p)$ are both p-conservative and since $s^a_t = s(A,p)^a_t$, we get $|s_t| = |s(A,p)_t|$. Case 1) is characterised by the properties $\{\tau_i,\tau_j\} \subset s_t$, $\tau_i \in s(A,p)_t$ and $\tau_j \in s(A,p)^a_t \setminus s(A,p)_t$. Then there exists τ_k which satisfies the properties $\tau_k \in s(A,p)_t$ and $\tau_k \notin s_t$. The existence of such a task leads to $(\tau_k,\tau_j) \in RP(A)$ and $(\tau_j,\tau_k) \in RP(s)$, that is incompatible with the hypothesis

$$\begin{cases} RP(s) \subset RP(A) \\ RP(A) \text{ does not contain cycles} \end{cases}$$

This is a contradiction: Case 1) is impossible.

Case 2

The proof is similar to that of Case 1): we obtain a task τ_k which satisfies $\tau_k \in s_t$ and $\tau_k \notin s(A,p)_t$. These properties lead to $(\tau_k,\tau_i) \in RP(s)$ and $(\tau_i,\tau_k) \in RP(A)$. We obtain a contradiction.

Case 3

This case leads to $s_t = s(A,p)_t$. Since both other cases cannot yield, Case 3 gives the only possible conclusion.

QED

Theorem 2: Let $s \in S_\Xi$ be a p-conservative sequence such that $RP(s)$ does not contain cycles. There exists a configuration $A \in \mathcal{C}_\Xi$ which satisfies $s(A,p) = s$. Moreover A is unique as soon as $RP(s)$ is a total relation.

Proof

$RP(s)$ does not contain cycle. Then it can be extended into a total and cycle-free relation R. Since $RP(s) \subset R$, there exists a fixed priority configuration A such that $RP(A) = R$.

To reach the result of Theorem 2, we proceed by induction on t to prove the property

$$\forall k \in \mathbb{N}^{\star}_{|\Xi|}, \forall (i,t) \in \mathbb{N}^2, C_{k,i,t}(s) = C_{k,i,t}(s(A,p))$$

For $t = 0$, this property yields.

Let us suppose that it stands for $t \in \mathbb{N}$. By definition of A, we get $s^a_{t+1} = s(A,p)^a_{t+1}$. Moreover, Lemma 2 proves that $s_{t+1} = s(A,p)_{t+1}$. Then, we get

$$\forall k \in \mathbb{N}^{\star}_{|\Xi|}, \forall i \in \mathbb{N}, C_{k,i,t+1}(s) = C_{k,i,t+1}(s(A,p))$$

Hence this property stands for $t \in \mathbb{N}$. Consequently, the following property also stands:

$$\forall t \in \mathbb{N}, s^a_t = s(A,p)^a_t$$

Then Lemma 2 gives $\forall t \in \mathbb{N}, s_t = s(A,p)_t$.

If A is a priority configuration which generates s, we get $RP(s) \subset RP(A)$. If $RP(s)$ is a total relation, we get $RP(s) = RP(A)$, hence A is unique.

<u>QED</u>

These properties are useful to characterize the set of relations that are compatible with a fixed priority policy.

Definition 11:

1) Every binary transitive relation on Ξ is called a priority relation on Ξ. We denote by BTR_Ξ the set of these relations.
2) Every binary transitive and acyclic relation on Ξ is called a fixed priority relation on Ξ. We denote by $BTAR(\Xi)$ the set of these relations.

C. Consistency

Theorems 1 and 2 give a necessary and sufficient condition to decide if a specific scheduling sequence can be generated on-line from a specific fixed priority configuration.

The scope of this section is to extend this result to priority relations, i.e. to give a necessary and sufficient condition useful to decide if a specific priority relation is compatible with a specific scheduling sequence. We call this property *consistency*.

A total relation corresponds to a priority configuration which generates a unique scheduling sequence. On the opposite, when it is not total, nothing can be deduced in terms of compatibility with priority driven on-line scheduling generation.

Example 6: Consider the following task system:

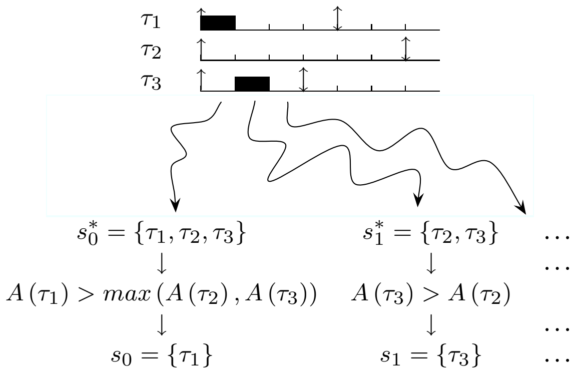
	r_i	C_i	D_i	T_i	$A(\tau_i)$
τ_1	0	1	4	4	3
τ_2	0	2	6	6	1
τ_3	0	1	3	3	2

We consider a specific priority configuration A which associates the priority $A(\tau_i)$ with task τ_i. Figure 3 presents the very first steps of the corresponding scheduling sequence. One can remark that scheduler decisions are deterministic (one possible choice at each switch time). Definition 10 gives the relation $RP(A)$ associated with A:

$$RP(A) = \{(\tau_1,\tau_2),(\tau_1,\tau_3),(\tau_3,\tau_2)\}$$

This relation leads the scheduling algorithm to produce the same sequence as A.

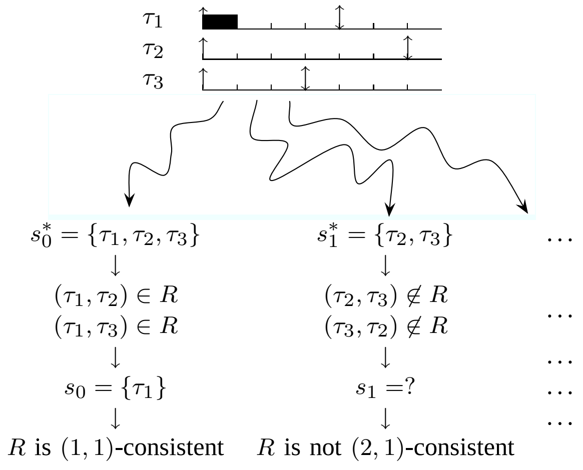
Now, consider the relation $R = \{(\tau_1,\tau_2),(\tau_1,\tau_3)\}$. Figure 4 presents the very first steps of the corresponding scheduling sequence. At time 0, R associates the highest priority with τ_1,



τ_1
τ_2
τ_3
$s^*_0 = \{\tau_1,\tau_2,\tau_3\}$
$A(\tau_1) > max(A(\tau_2), A(\tau_3))$
$s_0 = \{\tau_1\}$
$s^*_1 = \{\tau_2,\tau_3\}$
$A(\tau_3) > A(\tau_2)$
$s_1 = \{\tau_3\}$
...

Fig. 3. Generating a scheduling sequence from a priority relation computed from a fixed priority configuration

then τ_1 runs. At time 1, τ_1 is completed, then it is no more trying to access a processor. Consequently, the scheduler has to choose between τ_2 and τ_3. Unfortunately, R gives no information about the couple (τ_2,τ_3). Therefore, the scheduler can choose any of these tasks: R does not lead to a deterministic scheduler.



τ_1
τ_2
τ_3
$s^*_0 = \{\tau_1,\tau_2,\tau_3\}$
$(\tau_1,\tau_2) \in R$
$(\tau_1,\tau_3) \in R$
$s_0 = \{\tau_1\}$
R is $(1,1)$-consistent
$s^*_1 = \{\tau_2,\tau_3\}$
$(\tau_2,\tau_3) \notin R$
$(\tau_3,\tau_2) \notin R$
$s_1 = ?$
R is not $(2,1)$-consistent
...

Fig. 4. A scheduling sequence generated from a priority relation

Definition 12: Let $R \in BTAR(\Xi)$ and $t \in \mathbb{N}$. R is (t,p)-consistent if and only if there exists a (p,t)-conservative sequence $s \in S_\Xi$ such that $\bigcup_{u \in \mathbb{N}_{t-1}} RP_u(s) \subset R$.

A sequence s which satisfies the (p,t)-consistence property for R is called compatible with R.

We denote by $BTAR_{t,p}(\Xi)$ the set of (t,p)-consistent relations.

The useful properties of (t,p)-consistent relations are presented in the following lemmas. One can remark that the Lemma 2 leads $RP(A)$ to be (t,p)-consistent for all $t \in \mathbb{N}$ and all $p \in \mathbb{N}$ as soon as the configuration A concerns fixed priorities.

Lemma 3: Let $t \in \mathbb{N}$, $R \in BTAR_{t,p}(\Xi)$, and $s \in S_\Xi$ and $s' \in S_\Xi$ two R-compatible scheduling sequences. We get $u \in \mathbb{N}_{t-1} \Rightarrow s_u = s'_u$.

<u>*Proof*</u>

Since R is a (t,p)-consistent relation, it does not contain cycles. Thus, there exists a fixed priority configuration A such that $R \subset RP(A)$. Moreover, the lemma assumes $\bigcup_{t'=0}^{t'=t-1} RP_{t'}(s) \subset R \subset RP(A)$. Then we deal with the hypothesis of the Lemma 2, who requires $RP(s) \subset RP(A)$. Therefore, the Lemma 2 stands on

$\mathbb{N}_{t-1}$, and then we get

$$\forall t' < t, s^a_{t'} = s\left(A,p\right)^a_{t'} \Rightarrow s_{t'} = s\left(A,p\right)_{t'}$$

Using this property in the same way as in the proof of the Theorem 2, we obtain

$$\forall t' < t, s_{t'} = s\left(A,p\right)_{t'}$$

Following the similar way for s', we get

$$\forall t' < t, s'_{t'} = s\left(A,p\right)_{t'} = s_{t'}$$

QED

Therefore, all scheduling sequences compatible with a (t,p)-consistent relation share the same scheduling choices from time 0 to time t. This common subsequence is denoted $s\left(R,p,t\right)$. To satisfy the definition of scheduling sequences (see Definition 1), $s\left(R,p,t\right)$ must be precised for all $t' \in \mathbb{N}$, hence $s\left(R,p,t\right)_{t'}$ must be defined for $t' \geq t$. This is why we use the following extension:

$$\forall t' \geq t, s\left(R,p,t\right)_{t'} = \emptyset$$

Lemma 3 shows that consistency leads the sequence produced by the scheduler to be unique, and then the scheduler to be deterministic. The following lemma proves the monotonicity of consistency on $BTAR\left(\Xi\right)$.

Lemma 4: Let $t \in \mathbb{N}$, $R \in BTAR_{t,p}\left(\Xi\right)$ and $R' \in BTAR\left(\Xi\right)$. If $R \subset R'$, then R' is (t,p)-consistent and $s\left(R,p,t\right) = s\left(R',p,t\right)$.

Proof

By definition, $s\left(R,p,t\right)$ is compatible with R'. Therefore R' is (t,p)-consistent. By definition $s\left(R',p,t\right)$ is also compatible with R'. It follows that $s\left(R,p,t\right)$ and $s\left(R',p,t\right)$ are compatible with R' and the Lemma 3 involves

$$\forall t' < t, s\left(R,p,t\right)_{t'} = s\left(R',p,t\right)_{t'}$$

By definition we get also

$$\forall t' \geq t, s\left(R,p,t\right)_{t'} = s\left(R',p,t\right)_{t'} = \emptyset$$

QED

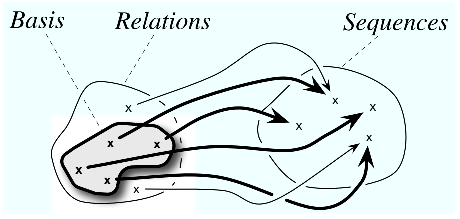
D. Basis for Consistent Relations

Searching valid fixed priority configurations leads to test all possible configurations (there are $|\Xi|!$ configurations to test). As soon as $|\Xi|$ is large, the enumerative approach can not be used anymore. In this section, we define a new concept, useful to limit the complexity of the testing process: the notion of basis of (t,p)-consistent relations (see Figure 5). These bases will allow us to determine the valid fixed priority configurations by avoiding the complete enumeration of configurations.

Definition 13: Let $t \in \mathbb{N}$ and $E \subset BTAR_{t,p}\left(\Xi\right)$. E is a basis of $BTAR_{t,p}\left(\Xi\right)$ if and only if for every $s \in S_\Xi$ which is compatible with a priority relation of $BTAR_{t,p}\left(\Xi\right)$, there exists a unique $R \in E$ such that $\forall t' < t, s\left(R,p,t\right)_{t'} = s_{t'}$.

Generating a basis is useful to avoid redundancy: the number of relations in E is the number of valid sequences of length t which can be generated from a fixed priority configuration.

There are several bases of $BTAR_{t,p}\left(\Xi\right)$. This property can be observed if $t = 0$. There is only one sequence s of length 0, so all bases of $BTAR_{0,p}\left(\Xi\right)$ contain exactly one element. Moreover, all fixed priority configurations generate the same sequence of length 0, and then each fixed priority configuration is a basis of



Generation
Every sequence is reached from the basis
Minimality
Avoiding one ↝ may lead a reached sequence to be lost
Avoiding one ↝ lets the set of reached sequences unchanged

Fig. 5. Characteristics of a basis of relations

$BTAR_{0,p}\left(\Xi\right)$. Lemma 4 leads to a special basis composed of *minimal* relations in the sense of inclusion.

Definition 14: Let $t \in \mathbb{N}$ and $R \in BTAR_{t,p}\left(\Xi\right)$. R is a minimal relation if and only if $R = RP\left(s\left(R,p,t\right)\right)$.

We denote by $B_{t,p}\left(\Xi\right)$ the set of minimal relations of $BTAR_{t,p}\left(\Xi\right)$.

This definition means that a minimal relation R is *optimal* for the generation of the sequence $s\left(R,p,t\right)$. The following theorem proves that $B_{t,p}\left(\Xi\right)$ is a basis of $BTAR_{t,p}\left(\Xi\right)$.

Theorem 3: Let $R \in BTR_\Xi$ and $t \in \mathbb{N}$. The two following assertions are equivalent:

- R is (t,p)-consistent
- $\exists! R' \in B_{t,p}\left(\Xi\right)$ such that $R' \subset R$

Proof

Suppose that there exists $R' \in B_{t,p}\left(\Xi\right)$ such that $R' \subset R$. By definition, R' is (t,p)-consistent, and Lemma 4 proves that R is also (t,p)-consistent, because $R' \subset R$.

Since R is (t,p)-consistent, we get $RP\left(s\left(R,p,t\right)\right) \subset R$. Moreover, this set is (t,p)-consistent and minimal. Then we get $RP\left(s\left(R,p,t\right)\right) \in B_{t,p}\left(\Xi\right)$. So, we get $RP\left(s\left(R,p,t\right)\right) \in B_{t,p}\left(\Xi\right)$ and $RP\left(s\left(R,p,t\right)\right) \subset R$.

Let $R' \in B_{t,p}\left(\Xi\right)$ such that $R' \subset R$. Lemma 4 proves that $s\left(R',p,t\right) = s\left(R,p,t\right)$, and we get (Definition 14) $R' = RP\left(s\left(R',p,t\right)\right)$. Therefore, we have $R' = RP\left(s\left(R,p,t\right)\right)$.

Hence for each minimal relation such that $R' \subset R$, we get $R' = RP\left(s\left(R,p,t\right)\right)$. Therefore, there exists a unique relation in $B_{t,p}\left(\Xi\right)$ such that $R' \subset R$.

QED

So the set $B_{t,p}\left(\Xi\right)$ is a basis of $BTAR_{t,p}\left(\Xi\right)$.

Definition 15: The set $B_{t,p}\left(\Xi\right)$ is called the minimal basis of $BTAR_{t,p}\left(\Xi\right)$.

The Theorem 3 shows that the minimal basis is useful to decide if a relation R is (t,p)-consistent: this property is reached as soon as R contains one relation of the basis, it is not necessary to go back to the corresponding scheduling. Note that the minimal basis is the sole basis that satisfies the Theorem 3.

E. Q-Validity

Previous sections provide results to decide if a specific scheduling sequence can be generated according to fixed priority rules. If this decision is positive, we know how to generate a fixed priority relation compatible with all fixed priority configurations useful to generate the sequence.

We are now interested in integrating usual specific real-time constraints into this methodology: deadlines, resource sharing, etc. This is the aim of this section: constraints are modeled by logical conditions, and the validation process is extended thanks to these conditions.

In that follows, Q denotes a time-dependent condition which concerns sequences: it collects all constraints to be satisfied by the sequences. We denote $Q_t(s)$ when the sequence s satisfies the constraints specified by Q at time t.

Definition 16: Let $s \in S_\Xi$.

- Let $t \in \mathbb{N}$. The sequence s is (Q,t)-valid if and only if $\forall t' < t, Q_{t'}(s)$.
- The sequence s is Q-valid if and only if $\forall t \in \mathbb{N}, Q_t(s)$.

The following definition extends the notion of basis to (Q,t)-valid sequences.

Definition 17: Let $t \in \mathbb{N}$. We call Q-minimal basis of $BTAR_{t,p}(\Xi)$ the set $B^Q_{t,p}(\Xi)$ of relations on Ξ defined in the following way:

$$R \in B^Q_{t,p}(\Xi) \Leftrightarrow \begin{cases} R \in B_{t,p}(\Xi) \\ s(R,p,t) \text{ is } (Q,t)\text{-valid} \end{cases}$$

Then, $B^Q_{t,p}(\Xi)$ is the subset of $B_{t,p}(\Xi)$ which collects all relations useful to generate (Q,t)-valid sequences. This new kind of basis is helpful to decide if a priority relation generates a valid scheduling sequence (in the sense of Q). This decision is reached thanks to a non-constructive process: only relations are used.

Theorem 4: Let $t \in \mathbb{N}$ and $R \in BTAR_{t,p}(\Xi)$. The two following assertions are equivalent:

1) $s(R,p,t)$ is (Q,t)-valid
2) $\exists! R' \in B^Q_{t,p}(\Xi)$ such that $R' \subset R$

Proof

Let $t \in \mathbb{N}$ and $R \in BTAR_{t,p}(\Xi)$. The Theorem 3 shows that there exists a unique $R' \in B_{t,p}(\Xi)$ such that $R' \subset R$, and the Lemma 4 proves that both R and R' generate the same sequence s.

$\boxed{(1) \Rightarrow (2)}$ Suppose that the sequence $s(R,p,t)$ generated by R is (Q,t)-valid. Then the sequence generated by R' is also (Q,t)-valid, and $R' \in B^Q_{t,p}$.

$\boxed{(1) \Leftarrow (2)}$ Suppose now that $R' \in B^Q_{t,p}$. Then $s(R',p,t)$ is (Q,t)-valid. Consequently, $s(R,p,t)$ is also (Q,t)-valid.

QED

This result proves that $B^Q_{t,p}(\Xi)$ is a basis of the subset of $BTAR_{t,p}(\Xi)$ composed of the sole relations compatible with a (Q,t)-valid sequence.

A fixed priority configuration corresponds to a total priority relation, so the Theorem 4 can be applied to fixed priority configurations.

F. Building the Minimal Basis

Figure 6 presents the fixed priority generation process. Suppose that we are in a valid state at time t, i.e. the fixed priority relation is able to schedule the software on $\mathbb{N}_{t-1}$). We know the (t,p)-consistent relation R and then the sequence $s(R,p,t)$. The problem is now to decide:

- if R remains consistent when extending the time interval to $\mathbb{N}_t$;
- if it satisfies the constraints modeled by Q on this interval.

We solve it thanks to a constructive process that follows a recursive iteration on t. Recall that we have at disposal a sequence

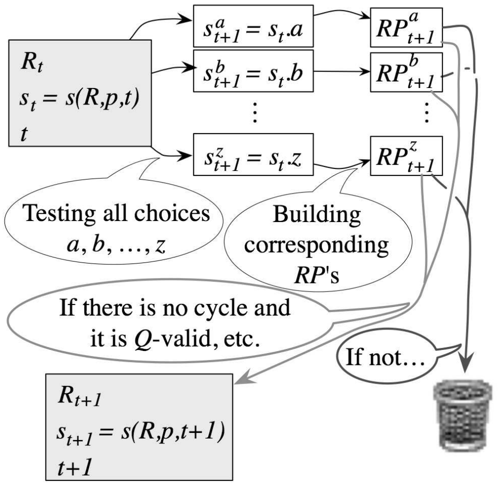


Fig. 6. Computing relations: from t to $t+1$

s which is valid up to time t: we have to extend s up to time $t+1$. We can do that by choosing (according to Q) the set α which collects all tasks to schedule at time t. s is a sequence of length $t-1$; the sequence $s.\alpha$, that extends s into a sequence of length t, is defined in the following way:

$$\begin{cases} u < t & \Rightarrow & (s.\alpha)_u = s_u \\ u = t & \Rightarrow & (s.\alpha)_u = \alpha \\ u > t & \Rightarrow & (s.\alpha)_u = \emptyset \end{cases}$$

Moreover, the priority relation we deal with at each step must satisfy the two following properties:

1) $s(R,p,t)$ is (Q,t)-valid and (t,p)-conservative,
2) R is (t,p)-consistent.

Property (1) can generally be decided for $s(R,p,t)$ thanks to the knowledge of the tasks running at time $t-1$. For Property (2), a solution consists in determining if R is an acyclic relation. This decision comes from Lemma 5.

Lemma 5: Let $R \in BTAR_{t,p}(\Xi)$ and $\alpha \subset s(R,p,t)^a_t$. The two following properties are equivalent:

1) The relation $\overline{R \cup RP_t(s(R,p,t).\alpha)}$ does not contain cycles, and $s(R,p,t).\alpha$ is $(p,t+1)$-conservative
2) $\begin{cases} |\alpha| = min\left\{p, \left|s(R,p,t)^a_t\right|\right\} \\ \forall (\tau_i,\tau_j) \in \alpha \times \left(s(R,p,t)^a_t \setminus \alpha\right), (\tau_j,\tau_i) \notin R \end{cases}$

Proof

Let $s = s(R,p,t)$. One can remark that $(s.\alpha)^a_t = s^a_t$ and $\overline{\cup_{t'=0}^{t-1} RP_{t'}(s)} = \overline{\cup_{t'=0}^{t-1} RP_{t'}(s.\alpha)}$.

$\boxed{(1) \Rightarrow (2)}$ The definition of conservativity involves $|\alpha| = min\left\{p, \left|s^a_{t+1}\right|\right\}$, hence the first condition of (2) is fulfilled.

On the one hand, we get (definition) $RP_{(s.\alpha)^a_{t+1}}(\alpha) = \left\{(\tau_i,\tau_j) | \tau_i \in \alpha \text{ and } \tau_j \in (s.\alpha)^a_{t+1} \setminus \alpha\right\}$. On the other hand, (1) assumes that $\overline{\cup_{t'=0}^{t+1} RP_{t'}(s.\alpha)}$ does not contain cycles. So, if $(\tau_i,\tau_j) \in RP_{(s.\alpha)^a_{t+1}}(\alpha)$, then $(\tau_j,\tau_i) \notin \overline{\cup_{t'=0}^{t} RP_{t'}(s.\alpha)}$. Hence the second condition of (2) is also satisfied.

$\boxed{(2) \Rightarrow (1)}$ We assume s to be (t,p)-conservative, and also $|\alpha| = min\{p, |s^a_t|\}$. Then $s.\alpha$ is $(t+1,p)$-conservative. From the second condition of (2), the (τ_j,τ_i)'s which correspond to a couple $(\tau_i,\tau_j) \in RP_{(s.\alpha)^a_{t+1}}(\alpha)$ do not belong to $\overline{\cup_{t'=0}^{t} RP_{t'}(s)}$. Moreover, we assume $\overline{\cup_{t'=0}^{t} RP_{t'}(s)}$ to contain no cycle. So no cycle can be found in $\cup_{t'=0}^{t+1} RP_{t'}(s.\alpha)$.

QED

The $(t+1,p)$-consistency leads all possible task sets α to contain the same number of tasks: $min\left\{p,\left|s_t^a\right|\right\}$. In general, there is no unicity for α. In the following, we denote by $SC_\Xi(R,p,t)$ the set of possible values for α.

All these results lead us to the following theorem, that gives a recursive method to build Q-minimal basis.

Theorem 5: The sequence $\left(B_{t,p}^Q\right)_{t\in\mathbb{N}}$ satisfies the following properties:

1) $t=0 \Rightarrow B_{t,p}^Q(\Xi)=\{\emptyset\}$
2) $t\geq 0 \Rightarrow$
$$B_{t+1,p}^Q(\Xi)=\bigcup_{R\in B_{t,p}^Q(\Xi)}\ \bigcup_{\substack{\alpha\in SC_\Xi(R,p,t)\\ Q_t(s(R,p,t).\alpha)}}\overline{R\cup RP_{s(R,p,t)_t^a}(\alpha)}$$

Proof

The relation $\emptyset$ is $(0,p)$-consistent and minimal. Then, we get $B_{0,p}^Q(\Xi)=\{\emptyset\}$.

Let $t\in\mathbb{N}$. We note

$$E=\bigcup_{R\in B_{t,p}^Q(\Xi)}\ \bigcup_{\substack{\alpha\in SC_\Xi(R,p,t)\\ Q_t(s(R,p,t).\alpha)}}\overline{R\cup RP_{s(R,p,t)_t^a}(\alpha)}$$

Let $R\in B_{t,p}^Q(\Xi)$, $\alpha\in SC_\Xi(R,p,t)$ and $R'=\overline{R\cup RP_{s(R,p,t)_t^a}(\alpha)}$.

By definition, we obtain $s\left(R',p,t+1\right)=\left(s(R,p,t).\alpha\right)$. From the hypothesis of Theorem 5 and Lemma 5, we get

- $RP\left(s(R,p,t).\alpha\right)$ does not contain cycles,
- $s(R,p,t).\alpha$ is $(p,t+1)$-conservative,
- $s(R,p,t).\alpha$ is $(Q,t+1)$-valid.

These properties yet stand for $s\left(R',p,t+1\right)$. We get $RP\left(s(R,p,t).\alpha\right)=R'$, and then R' is minimal. So $R'\in B_{t+1,p}^Q(\Xi)$.

Let now be $R\in B_{t+1,p}^Q(\Xi)$, and denote by R' the set $RP\left(s(R,p,t)\right)$. Then $R'\in B_{t,p}^Q$ and $s\left(R',p,t\right)=s(R,p,t)$. By hypothesis, $s(R,p,t+1)$ is $(p,t+1)$-conservative and $(Q,t+1)$-valid. From Lemma 5, we obtain $s(R,p,t+1)_{t+1}\in SC_\Xi\left(R',p,t\right)$, and therefore $RP\left(s(R,p,t+1)\right)\in E$. Since R is minimal, we have $R=RP\left(s(R,p,t+1)\right)$, and it follows that $R\in E$.

QED

This theorem gives a constructive technique for the minimal basis, that is useful to generate all fixed priority configurations which satisfy the constraint Q.

IV. Computing the priorities

In this section, we use Theorem 5 to produce all valid fixed priority configurations which are useful to schedule independant task systems. Firstly we specify the predicate functions (Q) corresponding to time constraints and secondly we give the stopping condition for the generation. Then, we give the algorithm.

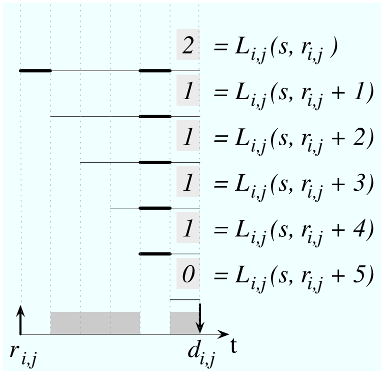
A. Validity Constraints

For independent tasks, valitidy constraints are limited to time constraints: each task must reach its deadline. In our approach, this property is expressed in the following way:

$$\forall\tau_i\in\Xi,\forall j\in\mathbb{N},C_{i,j}\left(s,d_{i,j}\right)=C_i$$

This property can be computed by the use of laxities (see Figure 7):

$$\forall\tau_i\in\Xi,\forall j\in\mathbb{N},L_{i,j}(s,t)=\left(d_{i,j}-t\right)-\left(C_i-C_{i,j}(s,t)\right)$$



2 = L_{i,j}(s, r_{i,j})
1 = L_{i,j}(s, r_{i,j} + 1)
1 = L_{i,j}(s, r_{i,j} + 2)
1 = L_{i,j}(s, r_{i,j} + 3)
1 = L_{i,j}(s, r_{i,j} + 4)
0 = L_{i,j}(s, r_{i,j} + 5)
r_{i,j}
d_{i,j}
t

Fig. 7. Dynamic laxity of a task

One can show that each task reaches its deadlines if and only if at any time, all tasks with null laxities are running. In the following, we denote by $V_t(s)$ this property.

B. Stabilizing $B_{t,p}^V(\Xi)$

The Theorem 5 gives a recursive constructive process to build the V-minimal basis. The aim of this section is to obtain the recursion stopping condition. In the following, we denote $\underset{\tau_i\in\Xi}{max}(r_i)$ by r, and $\underset{\tau_i\in\Xi}{lcm}(T_i)$ by P.

The uniprocessor sequences which are produced following fixed priority configurations are cyclic with period P, and the entrance into the cyclic behaviour is reached before the date $r+P$ [10]. So, in this context, the time interval to scan for computing the valid fixed priority configurations is limited to $[0,r+2P]$.

For synchronous systems ($\forall k\in[1\,|\Xi|]\,,r_k=0$), the multiprocessor sequences which are produced following fixed priority configurations are cyclic with period P, and the cyclic behaviour starts at date 0 [11]. Therefore, in this context, the time interval to scan for computing the valid fixed priority configurations is limited to $[0,P]$.

For these two contexts, the time interval to scan does not depend on the used fixed priority configuration. Then the basis $B_{M,p}^V(\Xi)$ collects all valid fixed priority configurations, respectively with

1) $M=r+2P$ for uniprocessor,
2) $M=P$ for synchronous task systems.

For asynchronous task systems ($\exists(i,j)\in[1,|\Xi|]^2$ such that $r_i\neq r_j$) which are scheduled on multiprocessor following a fixed priority configuration, [11] gives a feasibility interval. Let us consider the induction defined in the following way:

$$\begin{array}{l}S_1=r_1\\ \forall k\in[2,|\Xi|]\,,S_k=\max\left\{r_k,T_k.\left\lceil\frac{S_{k-1}-r_k}{T_k}\right\rceil\right\}\end{array}$$

If tasks are sorted by priorities ($i>j\Rightarrow priority(\tau_i)>priority(\tau_j)$), the multiprocessor sequence produced following these priorities reaches its cyclic behaviour before $S_{|\Xi|}$. However, since $S_{|\Xi|}$ depends on the priority configuration we deal with, computing a feasibility interval valid for all fixed priority configurations leads to consider the maximum of the $|\Xi|!$ different possible values for $S_{|\Xi|}$. Therefore the complexity level is high.

The results presented in [11] propose a feasibility interval, but they are also helpful for a cyclicity diagnosis onto sequences. Such results can be obtained observing the state of the task system. Following the sequence s, the state of task τ_k at time t is described by $C_{k,\left\lfloor\frac{t-r_k}{T_k}\right\rfloor}(s,t)$, and the state of the whole task system is the family of states of the individual tasks. Let us now consider a time $t \geq r+P$. If the task system returns at time t in the state they were at time $t-P$, then the sequence is cyclic from $t-P$, and then the feasibility interval is $[0,t]$. Scanning the condition $State\,(Task\ system, t) = State\,(Task\ system, t-P)$ while t increases is useful to detect the time t_0 when all sequences are cyclic: from this time, the basis $B^V_{t,p}(\Xi)$ is constant.

C. Algorithm

The algorithm presented in Figure 8 is a recursive implementation for the computing of $B^V_{t,p}(\Xi)$. The data structure used is a list of 3-tuples composed in the following way:

- $\boxed{R}$
  a priority relation,
- $\boxed{\mathcal{I} = (I_k)_{\tau_k \in \Xi}}$
  the value $C_{k,\left\lfloor\frac{t-r_k}{T_k}\right\rfloor}(s(R,p,t),t)$ of the current job of τ_k,
- $\boxed{\mathcal{M} = (M_k)_{\tau_k \in \Xi}}$
  the state of all tasks at the beginning of the current meta-period.

$\mathcal{M}$ gives the value of $\mathcal{I}$ at time $r + P\left\lfloor\frac{t-r}{P}\right\rfloor$. Note that the $\mathcal{I}_k$'s and t give a total information on the current state of the system: laxities $Lax_k(\mathcal{I},t)$, active tasks $Act(\mathcal{I},t)$, and so on, can be deduced directly.

```
$rec\left(t, R, I, M, B', a, k\right)$:
  $s := \sum_{k=1}^{k=|\Xi|} a_k$
  if $k > |\Xi|$ then
    if $s = min\{p, |Act(\mathcal{I},t)|\}$ then
      $valid := true$
      for all $\tau_i \in \Xi$ such that $a_i = 1$,
      for all $\tau_j \in Act(\mathcal{I},t)$ such that $a_j = 0$
        if $(\tau_j, \tau_i) \in R$ then
          $valid := false$
      if $valid = true$ then
        $B' := B' \cup \left(\overline{R \cup RP_{s(R,p,t)^a_t}(a)}, I + a, \mathcal{M}\right)$
  else
    if $s < p \wedge \tau_k \in Act(\mathcal{I},t)$ then
      $rec\left(t, R, I, M, B', (a_1 \ldots\ a_{k-1}, 1, a_{k+1} \ldots\ a_n), k+1\right)$
    if $Lax_k(\mathcal{I},t) > 0$ then
      $rec\left(t, R, I, M, B', (a_1 \ldots\ a_{k-1}, 0, a_{k+1} \ldots\ a_n), k+1\right)$
```

Fig. 8. $B^V_{t,p}(\Xi)$ computing algorithm

We can now present the minimal basis computing algorithm (see Figure 9). Each step consists firstly in initializing the $\mathcal{I}_k$'s associated with tasks which are awaking, and secondly in searching for the start time of the cycling behaviour. The basis is built by calling the algorithm presented in Figure 8. When the execution is completed, res contains the minimal basis.

This computation stands on two basic operations on priority relations: *searching* and *adding* a couple (τ_i, τ_j) in R (for *adding*, the properties of the algorithm ensure that the relation remains cycle-free). A powerful data structure must be developed to represent $(R, \mathcal{I}, \mathcal{M})$.

Some optimizations can be done. For instance, there is no evolution of R while the running tasks remain unchanged: this

```
$t := 0$
$B := \{(\emptyset, (0 \ldots\ 0), (-1 \ldots\ -1))\}$
$res := \emptyset$
while $B \neq \emptyset$ do
  for all $k$ such that $t \in r_k + T\mathbb{N}$ then
    for all $(R, \mathcal{I}, \mathcal{M}) \in B$ do
      $I_k := 0$
  if $t \in r + P\mathbb{N}$ then
    for all $(R, \mathcal{I}, \mathcal{M}) \in B$ do
      if $\mathcal{I} = \mathcal{M}$ then
        $res := res \cup R$
        $B := B \setminus \{(R, \mathcal{I}, \mathcal{M})\}$
      else $\mathcal{M} := \mathcal{I}$
  $B' := \emptyset$
  for all $(R, \mathcal{I}, \mathcal{M}) \in B$ do
    $rec\left(t, R, \mathcal{I}, \mathcal{M}, B', (0 \ldots\ 0), 1\right)$
  $B := B'$
  $t := t + 1$
```

Fig. 9. Minimal base computing algorithm

observation leads us to use *time jumping* instead of computing solutions for each possible switch time.

This validation process is also modular: we can study validity for a subset of the task systems, the relation R obtained concerns the sole tasks which belong to the concerned subset. In a second step, this relation R can be integrated into a validation study of the whole system by initializing B with $\{(R, (0 \ldots\ 0), (-1 \ldots\ -1))\}$.

D. From the priority relation to the configuration

The algorithm presented in Section IV-C builds the V-minimal basis, which collects all valid minimal relations. Here, we are interested in valid configurations. This is why we must now map these priority relations into priority configurations. Relations give information on task priorities. Computing a priority configuration from a priority relation consists in choosing numeric values compatible with the relation.

If the priority relation R is total, one of the tasks (let us call it τ, and note that it is unique) is the R-greatest element of Ξ: τ's priority is the higher priority: we set $Priority(\tau) := |\Xi|$. We consider now the restriction $\frac{R}{\Xi \setminus \{\tau\}}$ of R to $\Xi \setminus \{\tau\}$. $\frac{R}{\Xi \setminus \{\tau\}}$ is also total: the task τ' of maximal priority can be designed, and its priority set to $|\Xi| - 1$. This process is iterated until the set contains only one task, that is associated with priority 1. This method provides a priority configuration according to the priority relation with at most $O\left(|\Xi|^3\right)$ operations.

If the priority relation R is partial, then there are two tasks τ_i and τ_j such that $(\tau_i, \tau_j) \notin R$ and $(\tau_j, \tau_i) \notin R$. This situation means that the relation R correspond to more than one priority configuration. When this context yields we build two intermediate relations $R_1 = \overline{R \cup \{(\tau_i, \tau_j)\}}$ and $R_2 = \overline{R \cup \{(\tau_j, \tau_i)\}}$, and we proceed recursively. This procedure gives a set of total relations.

V. Experimentation Results

A. An example of validation

Let us consider the following system of tasks, that we study on a 2-processor architecture.

		τ_j						
		τ_1	τ_2	τ_3	τ_4	τ_5	τ_6	τ_7
τ_i	τ_1		$\times$	$\times$	$\times$		$\times$	$\times$
	τ_2			$\times$				
	τ_3							
	τ_4		$\times$	$\times$			$\times$	$\times$
	τ_5		$\times$	$\times$	$\times$		$\times$	$\times$
	τ_6		$\times$	$\times$				$\times$
	τ_7		$\times$	$\times$				

Legend A box is marked with $\times$ if (τ_i, τ_j) belongs to the relation.

Fig. 10. Priority relation for the example.

Task	r_i	C_i	D_i	T_i
τ_1	15	7	11	38
τ_2	47	1	8	38
τ_3	4	4	43	45
τ_4	17	8	13	19
τ_5	43	3	3	6
τ_6	22	8	11	19
τ_7	30	6	25	25

The classical fixed priority scheduling policies (RM and DM) do not schedule this task system. One can note that also classical dynamic priority scheduling policies (EDF and LLF) do not schedule them. Let us now use the here-presented method PFX. The minimal basis is produced thanks to the algorithm presented in Section IV-C. The computing is completed at $t = 17147$ and produces a single priority relation (denoted R below). The graph of this priority relation is presented in Figure 10.

Since the basis is not empty, this task system can be scheduled by a fixed priority based scheduler. We use the method described in Section IV-D to determine the fixed priority configurations corresponding to this relation. One can remark that this relation is not total, since τ_1 and τ_5 can not be separated. Therefore, the partial relation can be used to produce two total relations which correspond to two different sets of solutions. In our example, these two relations are:

$$\left| \begin{array}{l} R_1 = R \cup \{(\tau_1, \tau_5)\} \\ R_2 = R \cup \{(\tau_5, \tau_1)\} \end{array} \right.$$

The relations R_1 and R_2 are total, so we can now compute some corresponding priority configurations:

Task	R_1	R_2
τ_1	7	6
τ_2	2	2
τ_3	1	1
τ_4	5	5
τ_5	6	7
τ_6	4	4
τ_7	3	3

This example shows that our method produces solutions which are out of reach of all classical methods, including LLF. These solutions are obtained quickly, since the computing time for this task system is less than 0.1 second on an $2.5GHz$ Apple-G5 machine.

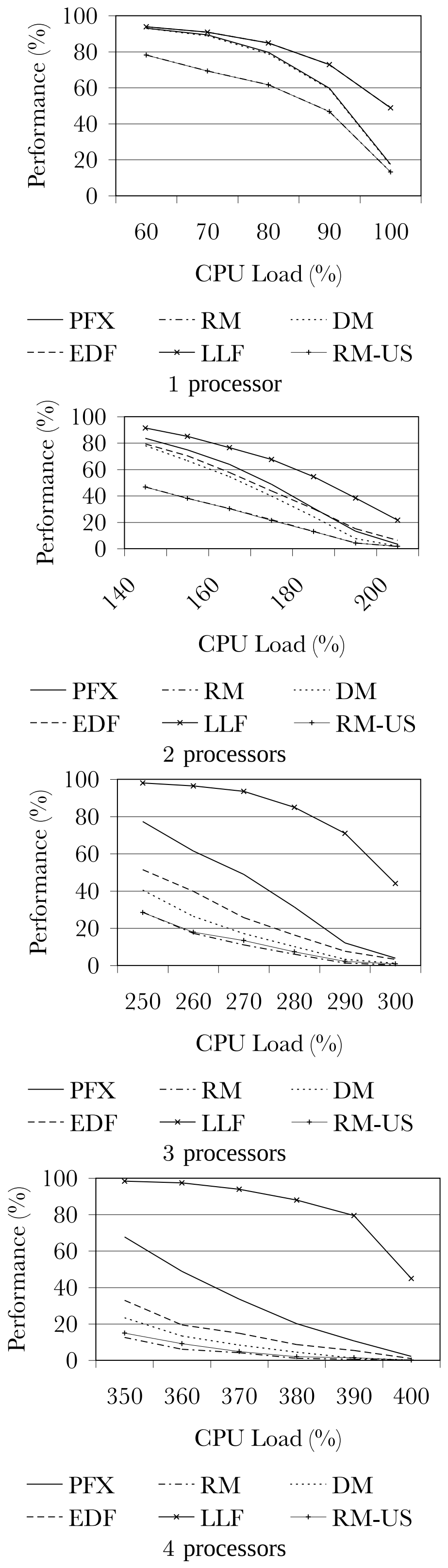


Fig. 11. Performances of each scheduling method

B. Average performances

In this section we compare our method PFX with the usual scheduling policies (RM, DM, EDF, LLF) and with the fixed priority scheduling method RM-US$\left[\frac{m}{3m-2}\right]$ proposed in [2]. We have generated samples composed of 1000 task sets. Each sample is characterized by a CPU-load and a target: all task systems which belong to a specific sample are designed for the same number of processors and share the same CPU-load. Task's parameters are integer numbers attributed by using the following rule:

1) the periods (T_k) are random numbers in $[1, 100]$,
2) the WCET (C_k) are random numbers in $[1, 40]$,
3) the deadlines (D_k) are random numbers in $[C_k, T_k]$,
4) the release times (r_k) are random numbersin $[0, T_k]$.

To obtain the required CPU-load for a task system, we add tasks until the targeted value is reached. Consequently, the number of tasks in a system is not fixed, but its average number is 10.

A scheduling simulation is performed for each generated task system, following all experimented scheduling techniques. So we associate each couple $(sample, scheduling\ technique)$ with the number of task systems in the sample which can be scheduled following the technique. We call this number *scheduling power* of the technique for the sample. Figure 11 presents the results of these experimentations.

We are interested in PFX performances, because it is the *fixed priority relation generation* technique presented in this work. PFX results correspond to the maximal scheduling power we can expect from the fixed priority schedulers existing in real-time kernels. One can remark that some generated task systems can not be scheduled in this way. EDF and LLF are optimal on uniprocessor, therefore their performances are references for our tests. For synchronous task systems, DM is also optimal on uniprocessor and consequently PFX does not bring a significative improvement. DM seems to be a very good fixed priority scheduling policy for the uniprocessor context.

On the contrary, no on-line policy is optimal in multiprocessor [12]. The evaluations presented in Figure 11 show that LLF outperforms all other methods. However, the schedules generated by LLF are characterized by large numbers of preemptions and switches. For some systems, this is not an issue; for others, a very high preemption level makes the resulting overhead unacceptable. In our performance evaluation computings, LLF must be view as an aid to minimize the number of task systems whose feasibility has to be tested.

Of course, PFX outperforms all fixed priority policies: we can observe that RM and DM policies do not reach the scheduling power of the fixed priority scheduling. We can also observe (and it is surprising) that PFX also outperforms EDF in the multiprocessor context. Last but not least: PFX can find all valid fixed priority configurations. This technique seems to be very promissing.

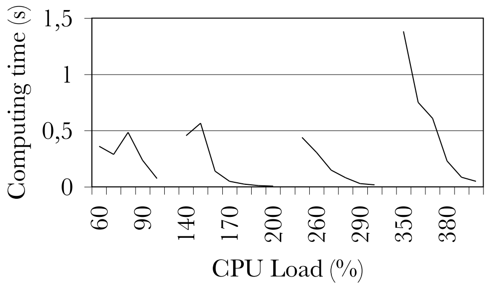


Fig. 12. PFX configurations computing time

Figure 12 presents the PFX solution average computing times for a task system. We deal with a 2.5GHz Apple-G5 machine. One can remark that computing times vary as the inverse of system loads. This is a consequence of using of minimal bases for representing sets of solutions at time t: the complexity is proportional to the number of valid sequences of length t instead of the number of fixed priority configurations. Hence, the more the systems are constrained, the less is the number of solutions to build. This (intuitive) property is very interesting, it certainly makes PFX a good solution for real case studies. Moreover, one can see our implementation of PFX as a breadth-first search. To get quickly a solution, one can implement it as a depth-first search.

VI. Concluding Remarks

We have presented a new method, PFX, for determining fixed priority configurations to schedule real-time systems on multiprocessor. Experimentations have shown that in this context:

1) RM and DM are powerless, since more than 50% of the fixed priority feasible task systems can not be scheduled using these policies,
2) PFX outperforms RM, DM, and also EDF,
3) PFX reaches solutions which can not be obtained with classical scheduling methods,
4) PFX can find **all** valid fixed priority configurations.

These results show that PFX is very useful and efficient for multiprocessor scheduling of real-time systems. Our implementation of PFX can be improved in different ways: optimizing data structure, depth-first search instead of breadth-first search, incremental search, etc. In a near future we plan to extend the scope of this technique with resource sharing protocols. Designing hybrid scheduling algorithms involving PFX and other classical scheduling techniques is also a challenging research topic which is in progress by the authors.

References

[1] B. Andersson, and J. Jonsson. Some insights on fixed-priority preemptive non partitioned multiprocessor scheduling. Real-Time Systems Symposium – Work-In-Progress Session, pp. 53–56, 2000.

[2] B. Andersson, S. Baruah, and J. Jonsson. Static-priority scheduling on multiprocessors. Real-Time Systems Symposium, pp. 193–202, 2001.

[3] N.C. Audsley, K. Tindell, and A. Burns. The end of the line for static cyclic scheduling? Euromicro Workshop on Real-time Systems, pp. 36–41, 1993.

[4] N.C. Audsley, A. Burns, R.I. David, K.W. Tindell, and A.J. Welling. Fixed priority preemptive scheduling: an historical perspective. Real-Time Systems, 8(2-3):173–198, 1995.

[5] T.P. Baker. An Analysis of Fixed-Priority Schedulability on a Multiprocessor. Real-Time Systems, 32(1-2):49–71, 2006.

[6] 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.

[7] S.K. Baruah, and J. Goossens. The static-priority scheduling of periodic task systems upon identical multiprocessor platforms. International Conference on Parallel and Distributed Computing and Systems, pp. 427–432, 2003.

[8] J. Carpenter, S. Funk, P. Holman, A. Srinivasan, J. Anderson, and S. Baruah. A Categorization of Real-Time Multiprocessor Scheduling Problems and Algorithms. In Handbook of Scheduling: Algorithms, Models, and Performance Analysis, Computer and Information Science Series, pp. 30–60, Chapman Hall/ CRC Press, 2004.

[9] A. Choquet-Geniet. Un premier pas vers l'étude de la cyclicité en environnement multi-processeur. Proceedings of Real-Time Systems, pp. 289–302, Paris, 2005.

[10] A. Choquet-Geniet and E. Grolleau. Minimal schedulability interval for real-time systems of periodic tasks with offsets. Theoretical Computer Science, 310:117–134, 2004.

[11] L. Cucu and J. Goossens. Feasibility Intervals for Fixed-Priority Real-Time Scheduling on Uniform Multiprocessors. International Conference on Emerging Technologies and Factory Automation (ETFA‘06), pp. 397–405, 2006.

[12] M.L. Dertouzos and A.K. Mok. Multiprocessor on-line scheduling of hard-real-time tasks. IEEE Transactions on Software Engineering, 15(12):1497–1506, 1989.

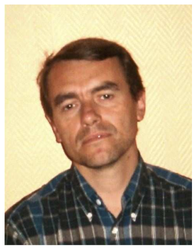
[13] J. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm : Exact characterisation and average case behaviour. Real-Time Systems Symposium, pp. 166–171, 1989.

[14] J.Y.-T. Leung and J. Whitehead. On the complexity of fixed priority scheduling of periodic real-time tasks Performance Evaluation, 2(4):237250, 1982.

[15] C.L. Liu and J.W. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. Journal of the ACM, 20(1):46–61, 1973.

[16] L. Sha, T.F. Abdelzaher, K.-E. Årzén, A. Cervin, T.P. Baker, A. Burns, G.C. Buttazzo, M. Caccamo, J.P. Lehoczky, and A.K. Mok. Real-time scheduling theory: A historical perspective. Real-Time Systems, 28(2-3):101–155, 2004.

Bernard Chauvière received his MS degree in computer science from University of Poitiers (France) in 2004. He is currently finishing his Doctoral degree at the *Laboratoire d'Informatique Scientifique et Industrielle* (Poitiers, France). His research interest is focused on real-time scheduling in multiprocessor.



Dominique Geniet received his MS degree and its PhD from University of Paris XI (France) respectively in 1986 and in 1989. He is Associate Professor at the University of Poitiers (France). His research interest is focused on real-time scheduling in multiprocessor.

René Schott has been a full Professeur at the University Henri Poincaré, Nancy (France), since 1987. His research interests include the study of stochastic processes and operator calculus on algebraic structures and their applications in computer science (probabilistic analysis of algorithms, optimization problems, real-time systems).