***L*aboratoire d'*I*nformatique *S*cientifique et *I*ndustrielle**

Rapport de recherche

N° 00 004

Maîtrise de la gigue temporelle avec les algorithmes d'ordonnancement DM et ED

L. David, F. Cottet, E. Grolleau
{david | cottet | grolleau} @ensma.fr

ECOLE NATIONALE SUPERIEURE DE MECANIQUE ET D'AEROTECHNIQUE
Site du Futuroscope – BP 40109 – 86961 FUTUROSCOPE Cedex – FRANCE
Tél : +33 (0) 5 49 49 80 63 – Fax : +33 (0) 5 49 49 80 64
Web : http://www.lisi.ensma.fr

Résumé :
L'utilisation d'un ordonnancement en ligne selon les algorithmes classiques (RM, DM et ED) ne permet pas au cours de l'exécution d'une application temps réel, de maîtriser le phénomène de gigue temporelle de certaines tâches. Cette gigue s'avère d'autant plus néfaste lorsque les tâches ont une période qui doit être strictement respectée (acquisition de données, commande d'un actionneur, etc.). Nous nous proposons donc dans ce travail de fournir une méthode permettant d'annuler la gigue temporelle pour des tâches dont les contraintes de régularité sont strictes, ceci dans le cadre des ordonnancements basés sur l'échéance (DM, ED).

Mots-Clés :
Gigue temporelle, ordonnancement en ligne, algorithmes d'ordonnancement.

TABLE DES MATIERES

I INTRODUCTION

Dans les systèmes temps réel à contraintes strictes, chaque tâche se doit de respecter ses contraintes temporelles. Le début d'exécution d'une instance d'une tâche doit, par exemple, commencer après sa date d'activation (date à laquelle la tâche demande la ressource processeur), la fin d'exécution doit, quant à elle, être réalisée avant l'échéance de l'instance. Dans le contexte d'ordonnancement de tâches temps réel à contraintes strictes dont les demandes d'exécution sont répétitives, le modèle classique de Liu et Layland [1] basé sur des tâches périodiques à 4 paramètres, ne suffit pas à caractériser les contraintes de temps de tous les systèmes temps réel, dans ce modèle seules les contraintes d'échéance de chaque tâche sont exprimées. Dans certains systèmes cependant, les tâches répétitives doivent satisfaire des contraintes de régularité d'exécution. Beaucoup d'exemples tournent autour des entrées/sorties de tels systèmes : le maintien de la stabilité d'un système de pilotage d'un procédé passe par une commande stable (au sens période de restitution) des actionneurs, les trames vidéos des systèmes multimédia doivent être transmises dans une durée bornée [2], la manipulation de matières dangereuses par des robots nécessite une bonne transmission de l'information entre le procédé et le système temps réel [3]. Un système dont la régularité d'exécution des tâches le composant est bien respectée, aura par ailleurs une tolérance aux fautes plus importante qu'un système supportant les irrégularités d'exécution : en effet, ce dernier devra attendre plus longtemps avant d'être sûr qu'un incident s'est produit (et cette détection plus rapide de la faute, et son traitement, pourront ainsi augmenter la tolérance aux fautes du système) [3].

Les périodes d'échantillonnage et/ou de restitution se doivent donc, sous certaines conditions et pour des besoins particuliers, d'être précisément respectées. Dans un contexte d'exécutions concurrentes comme le temps réel, ceci n'est pas toujours le cas. Dans la littérature, il existe de nombreux qualificatifs pour nommer cette irrégularité d'exécution. On parle de gigue temporelle, de gigue, de délais imprévisibles entre deux exécutions, de distances temporelles non constantes, etc [4] [5] [3]. Il faut distinguer en outre la gigue spécifique à l'exécution d'une tâche, et la gigue de bout en bout d'un système, celle-ci étant souvent en fait la variance du temps de réponse du système à un événement. Le temps de réponse de bout en bout d'un système informatique contient par ailleurs, le temps de réponse de la partie informatique avec donc, tous ses problèmes de respects des contraintes de temps, mais il contient aussi le temps de réponse du processus physique contrôlé (inertie thermique, réduction d'un couple moteur, etc…) qui s'adresse alors aux spécialistes du domaine.

Dans ce rapport, nous nous intéressons à la gigue d'exécution d'une tâche périodique. Elle mesure en fait l'irrégularité d'exécution d'une tâche par rapport à ses instances. Nous verrons par la suite une technique pour estimer ces irrégularités. Une méthode hors-ligne d'ordonnancement avec contrôle de la gigue dans le cas réparti a été présentée par DiNatale et Stankovic dans [5]. Une autre solution hors-ligne au problème du respect de la cadence d'échantillonnage, réside dans la recherche exhaustive des séquences valides respectant ce critère [6], [7]. Lin et al. proposent enfin un modèle dans lequel les contraintes de régularité d'exécution sont prises en compte, avec des algorithmes d'ordonnancement bien spécifiques [2] [3]. Nous nous intéressons ici au contexte des algorithmes d'ordonnancements en ligne (RM, DM et ED), pour lesquels nous proposons une méthode permettant de maîtriser la gigue des tâches à contraintes de régularité stricte, que nous appellerons indifféremment tâches régulières.

Nous nous proposons dans un premier temps de formaliser le calcul de la gigue temporelle pour une tâche périodique. Puis nous présentons une méthode de traitement de la gigue pour des tâches régulières de durée d'exécution unitaire, et nous généralisons enfin cette méthode aux tâches de durée quelconque.

Dans cette étude, nous nous plaçons dans l'hypothèse d'un environnement monoprocesseur sur lequel sont traitées des tâches périodiques et indépendantes. Nous utilisons le modèle de Liu & Layland [1] (cf. figure I.1), où chaque tâche est caractérisée par 4 paramètres temporels :

$r_{i,1}$: date de la première activation de la tâche,
C_i : durée d'exécution maximale,
D_i : délai critique,
P_i : période de la tâche.

Nous utilisons également les notations suivantes :
$r_{i,k}$: date de réveil de la $k^{ième}$ instance de la tâche i ($r_{i,k} = r_{i,1} + (k-1)P_i$),
$d_{i,k}$: échéance de la $k^{ième}$ instance de la tache i ($d_{i,k} = r_{i,k} + D_i$),
X^* : la nouvelle valeur d'une variable X après une modification, X peut être une date de première activation ou un délai critique.

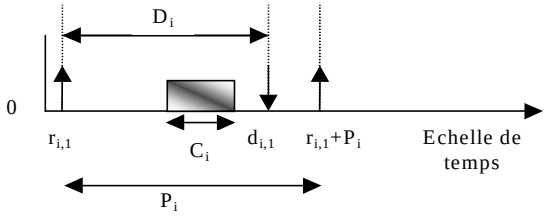

Figure I-1 *Modèle de tâches utilisé*

II LA GIGUE TEMPORELLE

Le phénomène de gigue temporelle est le terme employé en informatique temps réel pour définir l'irrégularité d'exécution d'une tâche périodique. Cette irrégularité peut être évaluée de plusieurs façons. Nous nous proposons ici de l'évaluer sur une période d'étude qui est donnée, dans le cas d'un système de tâches périodiques à départ différé, par [8], [9]:

$$P_{Etude} = \max_i(r_{i,1}) + 2\,ppcm(P_i) \qquad \textbf{(1)}$$

et se réduit à :

$$P_{Etude} = ppcm(P_i) \qquad \textbf{(2)}$$

pour un système de tâches périodiques à départ simultané.

Formalisons maintenant ce calcul : soient $s_{i,k}$ (respectivement $e_{i,k}$) les dates de début d'exécution (resp. de fin d'exécution) de la k^{ième} instance de la i^{ième} tâche d'une configuration de tâches périodiques (cf. figure III.2).

Les écarts entre deux débuts d'exécution (resp. entre deux fins d'exécution) concernant deux instances successives d'une tâche se définissent par :

$$\Delta s_{i,k} = s_{i,k+1} - s_{i,k} \quad (\text{resp. } \Delta e_{i,k} = e_{i,k+1} - e_{i,k})$$

où k désigne le k^{ième} écart sur la période d'étude. $\Delta s_{i,k}$ et $\Delta e_{i,k}$ varient entre les limites 0 et $2P_i$ (dans le cas de tâche à échéance sur requête avec des durées d'exécution négligeables).

Posons ensuite : $g_{S_{i,k}} = \dfrac{\left|\Delta s_{i,k} - P_i\right|}{P_i} 100\,\%$

qui évalue, en pourcentage par rapport à P_i, l'irrégularité des débuts d'exécution de la k^{ième} instance et de la k+1^{ième} instance :

- si l'écart vaut P_i (tâche régulière), alors $g_{S_{i,k}} = 0\,\%$.
- si l'écart tend vers 0 ou vers $2P_i$ (irrégularité maximum dans l'exécution), alors $g_{S_{i,k}} = 100\,\%$.

Définition II.1 *La moyenne arithmétique de ces termes nous permet alors de définir la gigue temporelle moyenne de la tâche T_i par :*

$$G_{moy}(T_i) = \frac{1}{N_i}\sum_{k=1}^{N_i} \frac{\left|\Delta s_{i,k} - P_i\right|}{P_i} 100\,\% \qquad \textbf{(3)}$$

où N_i représente le nombre d'intervalles sur la période d'étude, entre deux instances successives. N_i s'obtient par[1] *:*

$$N_i = \left\lceil \frac{P_{Etude}}{P_i} \right\rceil - 1$$

(On suppose P_{Etude} strictement plus grande que P_i.)

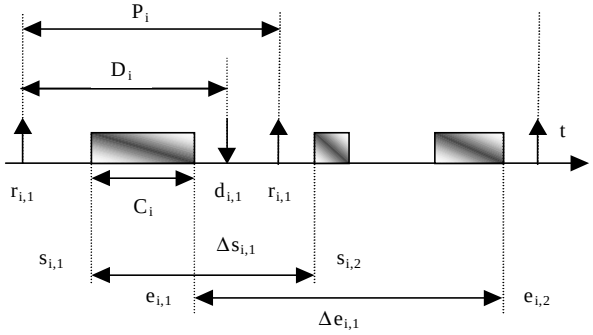

Figure II-1 *Définition des débuts et des fins d'exécution*

[1] $\lceil\ \rceil$ représente la partie entière supérieure.

Différentes études [10] [11] [12] ont montré les effets, parfois désastreux, de la gigue sur la qualité de transmission de l'information, au sens traitement du signal (échantillonnage, commande, etc…). Par ailleurs, les algorithmes d'ordonnancement RM, DM et ED reposent sur le modèle de Liu et Layland [1] où chaque exécution d'une tâche sur une période quelconque, est indépendante, en terme d'ordonnancement, de l'exécution de cette même tâche sur les autres périodes. Ainsi, le respect des contraintes de régularité de certaines tâches n'est pas pris en compte. Il est alors intéressant d'étudier de concert l'ordonnancement des tâches d'une configuration, ainsi que la maîtrise de la gigue temporelle de certaines tâches, tout cela, dans le contexte en ligne de RM, DM et ED.

Dans les configurations de tâches périodiques sur lesquelles nous travaillons, seules certaines sont considérées comme critiques en terme de respect strict de leur période, les autres sont périodiques sans contraintes strictes de régularité.

III METHODE DE TRAITEMENT DE LA GIGUE

Lors de la réalisation d'une application temps réel, le paramètre période P_i des tâches régulières est lié aux caractéristiques dynamiques du procédé, et ne peut donc pas être modifié. Le paramètre de durée d'exécution C_i est lui aussi supposé fixe car déterminé par le programme de la tâche. Seuls les paramètres $r_{i,1}$ et D_i peuvent, moyennant quelques précautions, être modifiés.

Le principe adopté pour diminuer la gigue, voire l'annuler, repose alors sur trois idées principales :
- 1/ décaler les $r_{i,1}$ des tâches régulières pour permettre des activations à des instants différents sur toute la période d'étude, et pour permettre à ces tâches de ne pas se concurrencer pendant leur exécution respective;
- 2/ rendre ces mêmes tâches, prioritaires par rapport aux autres non strictement régulières, pour que ces dernières ne viennent pas non plus perturber la régularité de l'exécution;
- 3/ vérifier l'ordonnançabilité de la nouvelle configuration si les délais critiques sont modifiés.

Notons que les périodes des tâches régulières peuvent être supposées toutes distinctes deux à deux, car si deux périodes s'avéraient identiques, il suffirait de définir une seule tache de même période et de durée d'exécution la somme des durées d'exécution de ces deux tâches.

Afin de simplifier le problème, nous nous intéressons dans une première approche aux tâches régulières de durée d'exécution unitaire, la durée unitaire correspondant à la granularité de l'ordonnanceur. Cette hypothèse est ensuite levée.

III.1 Tâches régulières de durée unitaire

III.1.1 Modification des dates d'activation:

Considérons 2 tâches régulières de durée unitaire : T_i et T_j. Pour ces deux tâches, nous devons éviter les perturbations à l'activation (cf. figure III.1).

Pour ces deux tâches, le décalage consiste à trouver un couple d'entiers naturels $(r_{i,1},r_{j,1})$ vérifiant la propriété suivante:

$$(\forall k \in \mathbb{N})(\forall k' \in \mathbb{N})\ r_{i,1} + k \cdot P_i \neq r_{j,1} + k' \cdot P_j \qquad \textbf{(4)}$$

Ce qui est équivalent mathématiquement (cf. annexe A) à trouver un couple d'entiers naturels $(r_{i,1},r_{j,1})$ de sorte qu'il n'existe aucune solution en x dans l'ensemble des entiers naturels au système :

$$S(i,j) \Leftrightarrow \begin{cases} x = r_{i,1} \bmod(P_i) \\ x = r_{j,1} \bmod(P_j) \end{cases}$$

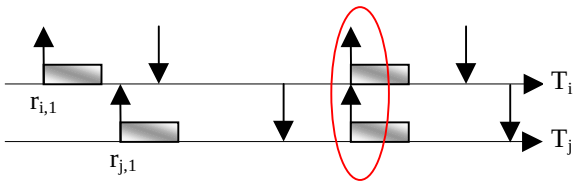

Figure III-1 - *Perturbation à l'activation de T_i par T_j*

En introduisant la notation mathématique modulo (mod) et le pgcd ($\wedge$), l'annexe B montre que le problème se résume à trouver un couple d'entiers naturels ($r_{i,1}$,$r_{j,1}$) vérifiant :

$$r_{j,1} - r_{i,1} \neq 0 \bmod(P_i \wedge P_j) \quad \text{où } P_i \text{ et } P_j \text{ sont nécessairement non premiers entre eux.}$$

Proposition III.1 *Une condition nécessaire pour la recherche des $r_{i,1}$ consiste à avoir tous les P_i non premiers entre eux deux à deux.*
(voir annexe B pour détails)

Ce résultat doit ensuite être étendu à l'ensemble des tâches à contraintes de régularité stricte. Soit n, le cardinal de cet ensemble, le décalage des activations de ces tâches consiste alors à trouver un n-uplet ($r_{1,1}$,$r_{2,1}$,…,$r_{n,1}$) vérifiant[2] :

$$\forall (i,j) \in [\![1;n]\!]^2 \text{ avec } i \neq j,\ r_{j,1} - r_{i,1} \neq 0 \bmod(P_i \wedge P_j) \qquad \textbf{(5)}$$

De cette condition générale (5), différentes méthodes peuvent être élaborées afin de trouver un ensemble de solutions. Ainsi nous avons les propositions suivantes:

Proposition III.2 *Si* $\min[P_i \wedge P_j] \geq n$ *alors il existe des solutions au problème posé. Dans ce cas, le n-uplet* $r_{1,1} = 0, \ldots, r_{n,1} = n-1$ *est solution ainsi que toute permutation de ces n nombres.*
(voir annexe C pour détails)

Proposition III.3 *Une condition suffisante pour la non-existence de solutions consiste à vérifier si* $\underset{i<j}{PPCM}(P_i \wedge P_j) \leq n-1$.
(voir annexe C pour détails)

Ce problème d'arithmétique est cependant complexe, et lorsque les outils analytiques démontrent l'existence d'une solution sans pouvoir la préciser (cf. annexe C pour d'autres propositions), nous faisons appel à des techniques d'énumération avec une recherche systématique de tous les $r_{i,1}$ possibles.

III.1.2 Changement de priorité

Nous devons ensuite rendre les tâches régulières prioritaires par rapport aux autres tâches de l'application. Cette action sur la priorité des tâches dépend essentiellement de l'algorithme d'ordonnancement choisi :
- en RM, la priorité étant attribuée en fonction de la période que nous considérons comme fixe, il n'est pas possible d'intervenir sur la priorité;
- en DM par contre, nous pouvons agir sur la priorité en modifiant les délais critiques. Soit T_i, une tâche à contrainte de régularité stricte. Afin d'imposer une priorité maximum à T_i par rapport aux tâches non régulières, nous choisissons un nouveau délai critique D_i^* tel que :

$$(C_i = 1) \leq D_i^* \leq \min_{j \neq i}(D_i, D_j - 1) \qquad \textbf{(6)}$$

Ainsi pour j désignant l'ensemble des tâches non nécessairement régulières, T_i a un délai critique strictement inférieur à tous les D_j, et se trouve donc prioritaire;
- en ED, il suffit d'imposer un nouveau délai critique tel que $D_i^*=C_i=1$.

III.1.3 Exemple d'application

Afin d'illustrer cette méthodologie, prenons l'exemple d'une configuration de 6 tâches dont deux sont des tâches d'acquisition ou de commande, de durée d'exécution unitaire, à contraintes de régularité. (cf. tableau III.1 pour commentaires)

Ces paramètres conduisent à une période d'étude de 72 unités de temps processeur, un taux d'utilisation processeur de 94,5 %, et des gigues temporelles que nous présentons dans le tableau III.2. Notons que si les tâches d'acquisition ou de commande devaient suréchantillonner le signal, la configuration ne serait pas ordonnançable.

[2] $[\![1;n]\!]$ représente l'ensemble des entiers compris entre 1 et n (inclus).

Tableau III-1 *Exemple de configuration de tâches régulières à durée d'exécution unitaire*

Tâches	$r_{i,1}$	C_i	D_i	P_i	Commentaires
Acquisition 1	0	1	8	8	Contrôle rapide (ex. contrôle d'ouverture d'une vanne)
Traitement 1	0	2	8	8	
Contrôle 1	0	1	7	8	
Acquisition 2	0	1	18	18	Mesure lente (ex. acquisition température)
Traitement 2	0	4	17	18	
Contrôle 3	0	1	6	6	Tâche de contrôle rapide (ex. alarme)

Tableau III-2 *Calcul des gigues sur les tâches régulières*

	RM	DM	ED
Gigue Acquisition1	9,4 %	9,4 %	10,9 %
Gigue Acquisition2	18,4 %	18.5 %	11.1 %

Nous appliquons alors la méthode présentée sur les trois algorithmes d'ordonnancement. Cette application est partielle dans le cas RM où nous n'intervenons pas dans la gestion de la priorité, contrairement aux deux autres cas DM et ED. La recherche des $r_{i,1}$ utilise la proposition III.2. Ainsi, en comparant les résultats obtenus à ceux présentés, le tableau III.3 illustre la limitation de la méthode dans le cas RM et la complète maîtrise de la gigue temporelle pour DM et ED. Nous pouvons remarquer que, dans le cas RM, le simple fait de décaler les dates d'activation modifie la gigue, mais permet aussi dans certains cas de la diminuer, voire de l'annuler. Les cas DM et ED fournissent, quant à eux, les résultats prévus : une gigue nulle.

Le nombre important de possibilités concernant le décalage pourrait permettre en outre de prendre en considération d'autres critères : le nombre de changements de contexte, le temps de réponse, etc…

Tableau III-3 - *Résultat du traitement d'annulation de la gigue des tâches d'acquisition*

	RM		DM	ED
	($r_{1,1}$=6, $r_{2,1}$=3)	($r_{1,1}$=3, $r_{2,1}$=2)	($r_{1,1}$=0, $r_{2,1}$=1) $1 \leq D_1^* \leq 5$, et $1 \leq D_2^* \leq 5$	($r_{1,1}$=0, $r_{2,1}$=1) et D_i^*=1
Gigue Acq1	8.1 %	0 %	0 %	0 %
Gigue Acq2	12.7%	15.9 %	0 %	0 %

III.2 Tâches de durée quelconque

Intéressons nous maintenant au cas des tâches dont les durées d'exécution sont plus grandes que la granularité choisie pour l'ordonnanceur. Dans ce cas, le simple décalage des dates de première activation ne suffit pas, il faut tenir compte de la durée d'exécution comme illustré sur la figure III.2. Par ailleurs, la propriété de préemptivité des tâches régulières ne va pas dans le sens d'une diminution de la régularité d'exécution.

Pour pallier ce problème, nous considérons que chaque tâche régulière est composée par une première partie à régularité stricte, celle correspondant à l'échantillonnage ou la commande, et par une seconde partie sans contrainte de régularité. Il est ensuite possible de découper cette tâche selon le principe de découpage en forme normale [13], [14], [15] en introduisant des contraintes de précédence artificielles, assurant un séquencement correct des deux parties de la tâche.

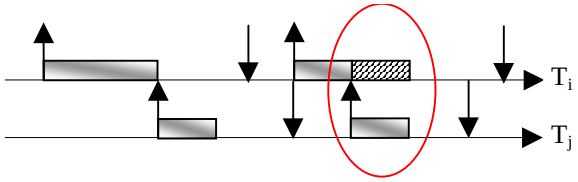

Figure III-2 *Perturbation entre deux tâches régulières dans le cas de tâches de durée quelconque : la zone hachurée de T_i rentre en conflit avec l'activation de T_j*

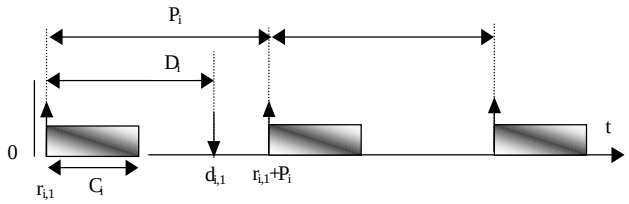

Figure III-3 *Exécution régulière et en bloc d'une tâche à contraintes de régularité stricte.*

Dans la suite, l'ensemble des tâches régulières est un ensemble de tâches pour lesquelles nous voulons une régularité stricte d'exécution, et pour lesquelles l'exécution se fait sans préemption (cf. figure III.3). La méthode employée pour traiter la gigue temporelle se présente alors en 4 points :
- 1/ décalage approprié des $r_{i,1}$, pour éviter les perturbations d'exécution entre les tâches régulières (exécution sans préemption),
- 2/ redéfinition des priorités des tâches régulières pour les rendre plus prioritaires que celles non nécessairement régulières,
- 3/ redéfinition des $r_{i,1}$ et, si besoin, des échéances pour les tâches qui sont liées par une contrainte de précédence artificielle, et ce, selon les méthodes de Blazewick [14] et Chetto [15],
- 4/ vérification de l'ordonnançabilité de la nouvelle configuration de tâches.

III.2.1 Modification des dates d'activation

Dans ces conditions, le choix des dates d'activations $r_{i,1}$ des tâches régulières est plus restrictif, et il faut prendre en compte la durée d'exécution C_i des tâches régulières dans la recherche des $r_{i,1}$. La proposition décrivant les conditions nécessaire et suffisante pour que deux tâches régulières devant s'exécuter sans préemption, ne puissent pas se concurrencer pendant leurs exécutions est décrite de façon détaillée dans l'annexe D :

Proposition III.3 *Soient deux tâches T_i et T_j non préemptibles, s'exécutant dès leur activation, de durée d'exécution respective C_i et C_j, et soit p leur PGCD. Ces deux tâches ne se perturbent pas entre elles pendant leur exécution si et seulement si*

$$\begin{pmatrix} r_{j,1}-r_{i,1} \neq 0 \bmod(p) \\ r_{j,1}-r_{i,1} \neq 1 \bmod(p) \\ \dots \\ r_{j,1}-r_{i,1} \neq C_i-1 \bmod(p) \end{pmatrix} et \begin{pmatrix} r_{i,1}-r_{j,1} \neq 0 \bmod(p) \\ r_{i,1}-r_{j,1} \neq 1 \bmod(p) \\ \dots \\ r_{i,1}-r_{j,1} \neq C_j-1 \bmod(p) \end{pmatrix}$$

Cette proposition peut facilement s'illustrer en disposant sur un cercle de périmètre $p = P_i \wedge P_j$, l'ensemble des valeurs interdites pour $r_{j,1}$–$r_{i,1}$ (cf. figure III.4).

Proposition III.4 *Avant toute recherche des $r_{i,1}$, il faut là encore vérifier que tous les P_i sont non premiers entre eux deux à deux.*
(voir annexe E pour détails)

Comme dans le cas précédent, il faut généraliser ce résultat à toutes les tâches régulières prises deux à deux. Posons pour cela, $S_{i,j}$ le système formé des deux systèmes ci-dessus. Il s'agit alors de trouver l'ensemble des $r_{i,1}$ des tâches à contraintes de régularité stricte répondant au problème suivant :

$$\forall (i,j) \in [\![1;n]\!]^2 \text{ avec } i \neq j \text{ , } (\, r_{i,1} \text{ , } r_{j,1} \,) \text{ solution de } S_{i,j}. \quad \textbf{(7)}$$

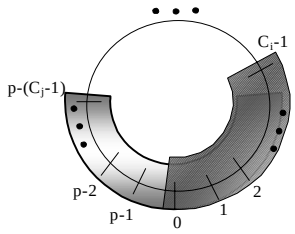

Figure III-4 *Les valeurs de $r_{j,1}$-$r_{i,1}$ interdites dans $\mathbb{Z}/p\mathbb{Z}$ sont grisées avec $p = P_i \wedge P_j$*

Proposition III.5 *La figure III.4. illustre une autre condition nécessaire d'existence de solutions au problème : pour tous les couples (i,j) d'entiers, avec i ≠ j, on doit vérifier* $P_i \wedge P_j > (C_j - 1) + (C_i - 1)$.

(voir annexe E pour détails)

Quelques résultats analytiques nous permettent là encore, dans certains cas de préciser des solutions [12], et dans les autres cas, nous effectuons une recherche des $r_{i,1}$ en utilisant des techniques d'énumération.

III.2.2 Le changement de priorité

Etudions maintenant la redéfinition des priorités :
- pour RM, pas de choix possible, car la priorité est fonction de la période que l'on suppose fixe,

- pour DM, même chose que dans le cas C_i=1, à ceci près que dans l'ensemble des tâches non nécessairement régulières, nous avons aussi celles qui sont liées aux tâches régulières par les contraintes de précédence. Rappelons cette condition :

$$C_i \leq D_i^* \leq \min_{j \neq i} (D_i, D_j - 1)$$

où j indice les tâches non nécessairement régulières et i les tâches régulières

D'un point de vue validation analytique, cela implique que les délais critiques des tâches non régulières doivent être plus grandes que les durées d'exécution des tâches régulières, soit : $\min(D_j) \geq \max(C_i)$.

- pour ED, on pose $D_i^* = C_i$, ce qui rend la tâche impérativement non préemptible sans quoi, une faute temporelle se produirait et la configuration ne serait pas ordonnançable. Ici aussi, nous devons prendre en compte les tâches qui sont liées aux tâches régulières par les contraintes artificielles de précédence. En pratique cependant, la configuration de tâches obtenue peut s'avérer non ordonnançable et il faut alors relâcher les contraintes au niveau des D_i^*. Une incrémentation progressive de ce paramètre est alors réalisée en prenant en compte de nouvelles dates d'activation. Effectivement, pour que la faute temporelle qui traduit la non ordonnançabilité de la configuration, puisse disparaître, il suffit d'autoriser la préemption de la tâche régulière, augmentant alors les contraintes pour la recherche des dates d'activation. La fin d'exécution de la tâche régulière est en effet repoussée (cf. figure III.5), et la technique de décalage de ce type de tâches en vue d'éviter les perturbations mutuelles, doit alors prendre en compte cette nouvelle fin d'exécution possible, de sorte que l'intersection des domaines possibles d'exécution des tâches régulières soit l'ensemble vide.

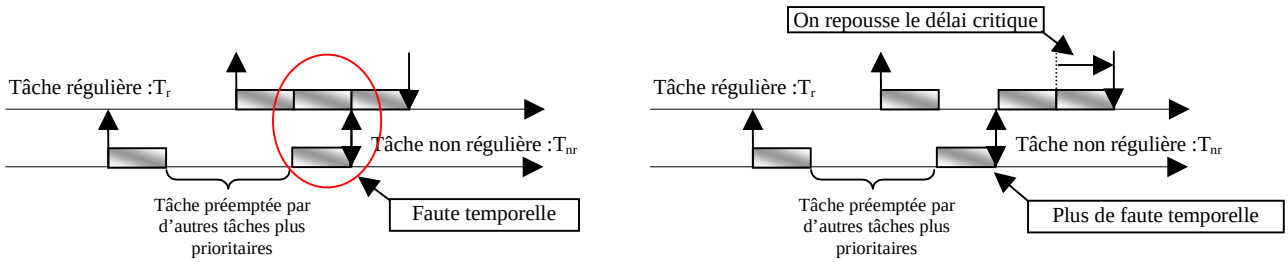

Figure III-5 – Augmentation du délai critique des tâches régulières pour éviter une faute temporelle.

Cette préemption peut n'avoir aucun effet sur la gigue telle que nous l'avons définie (cf. paragraphe II) si par exemple, elle se produit en milieu d'exécution de la tâche régulière, ou si par exemple la préemption décale l'ensemble des instances de la tâches de la même façon. Cependant, tel n'est pas le cas si la préemption se fait en début puis en fin d'exécution et ce, de manière alternée (cf. figure III.6).

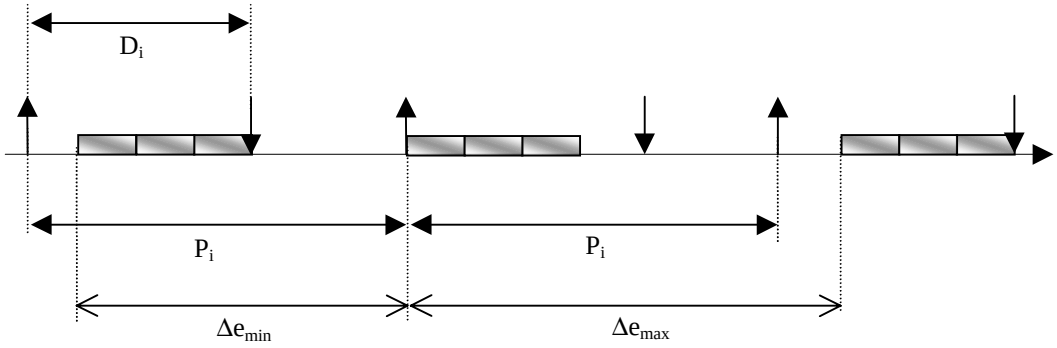

Figure III-6 – Configuration d'ordonnancement d'une tâche donnant une gigue maximum

Ainsi, dans le cas d'un relâchement des contraintes au niveau des délais critiques, on peut borner la gigue par une valeur maximale, qui représente le pire cas. Déterminons cette borne maximale, nous avons les relations suivantes :

$$\Delta e_{\max} = P_i + D_i^* - C_i$$
$$\Delta e_{\min} = P_i - (D_i^* - C_i)$$

qui conduisent alors à une gigue maximum pour la tâche T_i de :

$$G_{\max}(T_i) = \frac{D_i^* - C_i}{P_i}$$

III.2.3 Prise en compte des contraintes de précédence

Il reste ensuite à traduire la relation de précédence qui existe entre certaines tâches régulières et certaines tâches non nécessairement régulières. Selon [14][15], il suffit de définir de nouvelles dates d'activation et de nouvelles priorités. L'algorithme d'ordonnancement choisi conditionne les différentes méthodes. Prenons l'indice j pour les tâches non nécessairement régulières et l'indice i pour les tâches régulières. Pour toutes les tâches non nécessairement régulières ayant une contrainte de précédence, nous définissons :
- en DM, une nouvelle date d'activation

$$r_{j,1}^* = \max(r_{i,1}^*, r_{j,1}) \qquad \textbf{(8)}$$

où $r_{i,1}^*$ est la nouvelle date d'activation de la tâche régulière T_i, partie non préemptible, issue de la résolution de l'équation **(7)**. Quant à la priorité, nous n'avons pas besoin de la définir à nouveau, puisque les nouvelles priorités affectées aux tâches régulières sont, par construction, plus grandes que celles non nécessairement régulières.
- en ED, de la même façon, il suffit d'une part de prendre :

$$r_{j,1}^* = \max(r_{i,1}^* + C_i, r_{j,1}) \qquad \textbf{(9)}$$

et, d'autre part, de définir une nouvelle échéance tel que :

$$(d_{i,1}^*)^* = (D_i^* + r_{i,1}^*)^* = \min(D_i^* + r_{i,1}^*, d_{j,1} - C_j) \qquad \textbf{(10)}$$

où D_i^* et $r_{i,1}^*$ sont respectivement le nouveau délai critique et la date de première activation de la tâche T_i, issus des premières étapes de la méthode.

Notons que cette redéfinition de l'échéance contraint le paramètre de la même façon que dans le paragraphe sur la priorité : il ne s'agit pas d'augmenter le paramètre d'un côté, pour le diminuer de l'autre. Il est à noter également que la redéfinition de ces paramètres va dans le sens de la caractérisation de l'intervalle d'exécution minimum possible pour la tâche considérée, s'approchant de facto des critères de validation analytiques. Ainsi, la condition (9) de Blazewick s'avère trop drastique et pourrait être remplacée par la condition suivante :

$$r_{j,1}^* = \max(r_{i,1}^*, r_{j,1}).$$

En effet, par construction, Priorité(T_i)>Priorité(T_j), donc il suffirait d'imposer la condition $r_{j,1}^* \geq r_{i,1}^*$, pour respecter le bon séquencement des deux parties de la tâches.

III.2.4 Exemple d'application

Appliquons maintenant la méthode à un exemple de configuration de tâches (cf. tableau III.4). Deux sont à contraintes de régularité stricte, et nous décidons de rendre non préemptibles leurs deux premières unités de temps pro-

cesseur. Ces paramètres conduisent à une période d'étude de 224 unités processeurs, un taux d'utilisation processeur de 89%, et nous obtenons les gigues temporelles du tableau III.5.

L'application de la méthode sur ce système de tâches a nécessité l'emploi d'une technique d'énumération de solutions concernant le décalage, et nous a conduit à scinder les deux tâches d'acquisition, pour donner finalement la configuration du tableau III.6 pour laquelle la période d'étude est de 451, et pour laquelle il existe deux contraintes artificielles de précédence.

Tableau III-4 *Exemple de configurations de tâches*

Tâches	$r_{i,1}$	C_i	D_i	P_i
Acquisition 1	0	3	16	16
Traitement 1	0	4	14	14
Contrôle 1	0	1	13	14
Acquisition 2	0	3	32	32
Traitement 2	0	4	31	32
Contrôle 3	0	1	8	8

Tableau III-5 *Gigue des tâches à contraintes de régularité pour le cas des tâches de durée quelconque*

	DM	ED
Gigue Acq1	9,6 %	9,6 %
Gigue Acq2	12,5%	12 %

La redéfinition des priorités en DM ne pose pas de problème et il en est de même pour ED dans cet exemple. Pour les deux tâches régulières, Acquisition 1.1 et Acquisition 2.1, on définit :

- en DM, $(D_1^* , D_2^*)=(7 ,7)$,
- en ED, $(D_1^* , D_2^*)=(2 ,2)$.

Choix qui permet d'annuler la gigue des deux tâches d'acquisition dans les conditions citées.

Tableau III-6 *Configuration modifiée après décalage*

Tâches	$r_{i,1}$	C_i	D_i	P_i
Acquisition 1.1	**0**	**2**	**16**	**16**
Acquisition 1.2	**0 (DM) / 2 (ED)**	**1**	**16**	**16**
Traitement 1	**0**	**4**	**14**	**14**
Contrôle 1	**0**	**1**	**13**	**14**
Acquisition 2.1	**3**	**2**	**32**	**32**
Acquisition 2.2	**3 (DM) / 5 (ED)**	**1**	**32**	**32**
Traitement 2	**0**	**4**	**31**	**32**
Contrôle 3	**0**	**1**	**8**	**8**

Dans cette partie, pour les besoins de la méthode, nous avons défini des contraintes artificielles de précédence entre des tâches régulières s'exécutant en bloc et des tâches non nécessairement régulières. Cette démarche est donc généralisable à une configuration de tâches périodiques comportant des relations de précédence, à la seule condition que les tâches régulières ne soient précédées d'aucune tâche, de sorte que les conditions de Chetto et de Blazewick sur les $r_{i,1}$ ne viennent pas s'opposer à la redéfinition des $r_{i,1}$ par notre méthode.

IV CONCLUSION

Dans le cadre des applications temps réel comprenant certaines tâches à contrainte de régularité d'exécution stricte, nous proposons une méthode permettant de conserver les ordonnancements temps réel par échéance (DM, ED). En effet, ces algorithmes d'ordonnancement, qui facilitent la validation temporelle d'une application (soit par des vérifications analytiques, soit par des simulations) ne permettent pas de contrôler la périodicité stricte d'exécution des tâches. Afin d'aller dans ce sens, la méthode décrite se décompose en plusieurs phases : décalage des dates d'activations au travers des $r_{i,1}$, redéfinition des priorités au travers des D_i, et enfin, vérification de l'ordonnançabilité. Dans le cas des tâches indépendantes à durée d'exécution unitaire (granularité de l'ordonnanceur), nous obtenons alors des tâches avec une gigue temporelle nulle. Lorsqu'il s'agit de tâches de durée d'exécution non unitaire, les résultats sont plus restrictifs, seul l'algorithme DM permet d'appliquer la méthode et d'obtenir de façon sûre des gigues nulles. Dans le cas ED, un relâchement progressif des contraintes liées aux priorités doit souvent être mis en place pour obtenir la périodicité stricte des contraintes.

Les configurations de tâches dépendantes avec ou sans partage de ressources, ainsi que le cas distribué constitueront la suite logique de ce travail. Enfin dans une prochaine étape, plutôt que d'annuler la gigue totalement, nous essaierons

de généraliser la technique évoquée en III.2.2 pour spécifier une gigue maximum. Enfin, il reste à approfondir l'application des conditions de Blazewick décrite dans III.2.3 où des contraintes plus souples, nous l'avons vu sur un exemple, pourraient être envisagées.

BIBLIOGRAPHIE

1. Liu, C.L. and J.W. Layland, *Scheduling algorithms for multiprogramming in real-time environment.* Journal of the ACM, 1973. 20(1): p. 46-61.
2. Han, C.-C., K.-J. Lin, and C.-J. Hou, *Distance-constrained scheduling and its applications to real-time systems.* IEEE Transactions on Computers, 1996. 45(7): p. 814-826.
3. Lin, K.-J. and A. Herkert. *Jitter Control in Time-Triggered Systems*. in: *Proc. of the 29th Annual Hawaii International Conference on System Sciences*. 1996.
4. Buttazzo, G.C., *Periodic task scheduling*, in *Hard real-time computing systems*. 1997, Kluwer Academic Publishers. p. p. 77-108.
5. DiNatale, M. and A. Stankovic. *Applicability of Simulated Annealing Methods to Real-Time Scheduling and Jitter Control*. in: *Proc. of Real-Time Systems Symposium*. 1995. Pisa (Italy).
6. Cheng, S.-T. and C.-M. Chen. *A Cyclic Scheduling Approach for Relative Timing Requirements*. in: *Proc. of the 3rd IEEE Real-Time Applications Worshop*. 1996.
7. Choquet-Geniet, A., D. Geniet, and F. Cottet. *Exhaustive Computation of the Scheduled Task Execution Sequences of a Real-Time Application*. in: *Proc. of the 4th International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems*. 1996. Uppsala, Sweden.
8. Leung, J. and M. Merrill, *A note on preemptive scheduling of periodic real-time tasks.* Information Processing Letters, 1980. 11(3): p. 115-118.
9. Grolleau, E., A. Choquet-Geniet, and F. Cottet. *Cyclicité des ordonnancements au plus tôt des systèmes de tâches temps réel*. in: *Proc. of RenPar'10*. 1998. Strasbourg.
10. Balakrishnan, A.V., *On the problem of time jitter in sampling.* I.R.E. Trans. On Inf. Theory, 1962. vol. IT8: p. p. 226-236.
11. Jerri, A.J., *The Shannon Sampling Theorem - Its Various Extensions and Applications: A tutorial Review.* Proceedings of the IEEE, Nov 1977, 1977. 65(11): p. 1565-1596.
12. Castanié, F. *An approximately unbiased recovery of discret Fourier Transform altered by jittered sampling epochs*. in: *Proc. of Proc. of Intern. Conf. on Acoustic, Speech and Signal Processing (I.S.A.S.S.P.)*. 1982.
13. Babau, J.P. and F. Cottet, *Méthodologie temporelle des applications temps réel à contraintes strictes.* JESA, 1998. 32(5, 6): p. 595.
14. Blazewick, J., *Scheduling dependent tasks with different arrival times to meet deadlines*, in *Modelling and Performance Evaluation of Computers Systems*. 1976, North-Holland Publishing Company.
15. Chetto, H., M. Silly, and T. Bouchentouf, *Dynamic scheduling of real-time task under precedence constraints.* Real-time Systems, 1990. 2: p. 181-194.
16. Knuth, D.E., *Seminumerical algorithms*. 2nd edition ed. The Art of Computer Programming. Vol. vol. 2. 1981: Addison Wesley. 689 pages.

Annexe A – Equivalence des problèmes

$(\forall k \in N)(\forall k' \in N) r_i + k \cdot P_i \neq r_j + k' \cdot P_j$

$\Leftrightarrow$ (il n'existe pas de x entier naturel solution de $\begin{cases} x = r_i \bmod(P_i) \\ x = r_j \bmod(P_j) \end{cases}$).

Preuve :

Nous avons les équivalences suivantes :

$(\forall k \in N)(\forall k' \in N) r_i + k \cdot P_i \neq r_j + k' \cdot P_j$

$\Leftrightarrow \neg((\exists k \in N)(\exists k' \in N) r_i + k \cdot P_i = r_j + k' \cdot P_j)$

où le signe $\neg$ représente la négation de l'assertion entre parenthèses

$\Leftrightarrow \neg($ il existe x entier naturel solution de $(\exists k \in N)(\exists k' \in N)\begin{cases} x = r_i + k \cdot P_i \\ x = r_j + k' \cdot P_j \end{cases})$

$\Leftrightarrow \neg($ il existe x entier naturel solution de $(\exists k \in Z)(\exists k' \in Z)\begin{cases} x = r_i + k \cdot P_i \\ x = r_j + k' \cdot P_j \end{cases})$

Démo de l'équivalence précédente :

$\Leftarrow$ évident : si $k \in N$ alors $k \in Z$, donc par contraposée, nous avons cette implication.

$\Rightarrow$ supposons maintenant qu'il existe x entier naturel solution de $(\exists k \in Z)(\exists k' \in Z)\begin{cases} x = r_i + k \cdot P_i \\ x = r_j + k' \cdot P_j \end{cases}$, supposons que k et k' soient négatifs, alors en prenant un k_0 entier naturel assez grand, on a alors :

$\begin{cases} x + k_o \cdot P_i \cdot P_j = r_i + (k + k_0 \cdot P_j) \cdot P_i \\ x + k_o \cdot P_i \cdot P_j = r_i + (k' + k_0 \cdot P_i) \cdot P_j \end{cases}$ avec $k + k_0 \cdot P_j$ et $k' + k_0 \cdot P_i$ entiers naturels. Il existe donc x' entier naturel solution de $(\exists k_1 \in N)(\exists k_1' \in N)\begin{cases} x' = r_i + k_1 \cdot P_i \\ x' = r_j + k_1' \cdot P_j \end{cases}$, et par contraposée, nous avons prouvé l'implication.

$\Leftrightarrow \neg($ il existe x entier naturel solution de $\begin{cases} x = r_i \bmod(P_i) \\ x = r_j \bmod(P_j) \end{cases})$

$\Leftrightarrow$ (il n'existe pas de x entier naturel solution de $\begin{cases} x = r_i \bmod(P_i) \\ x = r_j \bmod(P_j) \end{cases}$).

Annexe B – Problème d'arithmétique

Le problème initial consiste à trouver un couple d'entiers naturels (r_1,r_2) de sorte qu'il n'existe aucune solution en x dans l'ensemble des entiers naturels au système :

$$S(1,2) \Leftrightarrow \begin{cases} x = r_1 \bmod(P_1) \\ x = r_2 \bmod(P_2) \end{cases}$$

Caractérisons ce ou ces couple(s) d'entiers.

L 'étude montre qu'il faut distinguer deux cas :

si $P_1 \wedge P_2=1$

Par le théorème de Bezout: $\exists(u,v)\in Z^2$ tel que $uP_1+vP_2=1$ et alors $\forall(r_1,r_2)\in N^2$ fixés, le système S(1,2) admet la solution sur Z :$x=r_2uP_1+r_1vP_2$ et quitte à rajouter plusieurs fois P_1P_2 on trouve une solution à S(1,2) sur N.

si $P_1\wedge P_2=d\geq 2$

Par le théorème de Bezout: $\exists(u,v)\in Z^2$ tel que $uP_1+vP_2=d$. L'étude montre qu'il y a encore deux cas à distinguer:

* si $r_1=r_2 \bmod(d)$:

Soit $r_1=r_2$ auquel cas $x=r_1+P_1P_2=r_2+P_1P_2$ est solution du système S(1,2).

Soit $r_1 \neq r_2$ et alors par le théorème de Bezout: $\exists(u,v)\in Z^2$ tel que $uP_1+vP_2=d$ et donc $\exists(u',v')\in Z^2$ tel que $u'P_1+v'P_2=r_1-r_2$ puisque $r_1-r_2|d$ et alors $x=r_1+u'P_1=r_2+v'P_2$ est solution du système S(1,2) et quitte à rajouter plusieurs fois P_1P_2 on trouve une solution à S(1,2) sur N.

* si $r_1 \neq r_2 \bmod(d)$

Supposons qu'il existe une solution x sur N de S(1,2) alors $\exists(k,k')\in Z^2$ tels que

$$\begin{cases} x = r_1 + kP_1 \\ x = r_2 + k'P_2 \end{cases}$$

et ainsi $r_1-r_2=kP_1-k'P_2$ or $d \mid (kP_1 - k'P_2)$ donc $d \mid (r_1 - r_2)$ ce qui contredit l'hypothèse et donc il n'existe pas de solution au système S(1,2).

$\Rightarrow$ Ainsi les seuls cas qui répondent au problème posé sont les cas où $P_1 \wedge P_2 \neq 1$ et les solutions sont les couples (r_1,r_2) tels que $r_1 \neq r_2 \bmod(d)$.

Annexe C – Quelques résultats

Rappelons que nous recherchons un n-uplet $(r_{1,1},r_{2,1},\ldots,r_{n,1})$ vérifiant :

$$\forall (i,j)\in [\![1;n]\!]^2 \text{ avec } i\neq j,\ r_{j,1}-r_{i,1}\neq 0 \bmod (P_i\wedge P_j) \quad \textbf{(5)}$$

où les « P_i » sont non premiers entre eux deux à deux.

Pour ne pas alourdir les notations cependant, nous utilisons dans cette annexe le n-uplet $(r_1,\ldots,r_n)$ à la place du n-uplet $(r_{1,1},r_{2,1},\ldots,r_{n,1})$.

Introduisons quelques notations pour les preuves qui vont suivre :

La notion de congruence:
On dit que deux entiers A et B sont congrus à un entier naturel C si il existe un entier relatif K tel que A=B+KC. On l'écrit A=B mod(C) et on le lit A=B modulo C. La relation de congruence est une relation d'équivalence, on parle donc de classes d'équivalence et plus couramment de classes modulo C .On note $\mathbb{Z}/C\mathbb{Z}$ l'espace des classes modulo C et ainsi, dire que A=B mod(C) est équivalent à dire que $\overline{A}=\overline{B}$ dans $\mathbb{Z}/C\mathbb{Z}$.

La notation d'ensemble d'entiers:
Soit A et B deux entiers tels que A<B on note $[\![A,B]\!]$ l'ensemble des entiers compris, au sens large, entre A et B.

La notation du PGCD:
Soit A et B deux entiers, on note $A\wedge B$ le PGCD de A et B.
On posera pour deux périodes P_i et P_j, $d_{ij}=P_i\wedge P_j$

Proposition C.1: si $\alpha=\min_{i<j} P_i\wedge P_j\geq n$ alors il existe des solutions au critère (5):

Preuve: Posons $d_{ij}=P_i\wedge P_j$ et montrons que $r_1=0,..,r_n=n-1$ est alors solution: en fait pour qu'un n-uplet soit solution il suffit que tous ses r_k soient compris entre 0 et $\alpha-1$ et qu'ils soient tous différents car alors leurs classes respectives modulo d_{ij} seront toutes distinctes et ce pour tout couple (i,j).

Dans le cas choisi les classes de tous les r_k modulo d_{ij} avec $\begin{cases}(i,j)\in [\![1,n]\!]^2, i\neq j\\ k\in [\![1,n]\!]\end{cases}$ sont toutes deux à deux distinctes (car les nombres r_k se succèdent et ils sont au nombre de n qui est inférieur à d_{ij}). Le n-uplet (0,..,n-1) vérifie donc le critère (5).

Nous obtenons ainsi un minimum de A_α^n solutions, toutes entièrement connues. De plus si tous les d_{ij} sont égaux entre eux alors ces A_α^n n-uplets solutions sont les seules solutions au critère (5).

Remarque : la condition de la proposition C.1 n'est pas nécessaire à l'obtention de solutions au critère (5), elle est juste suffisante. Par exemple pour n =4 et $P_1=84,P_2=6,P_3=21$ et $P_4=12$ il vient $d_{12}=6,d_{13}=21$, $d_{14}=12,d_{23}=3,d_{24}=6$ et $d_{34}=3$, et alors on ne vérifie pas la condition de la proposition C.1 bien que (0,1,2,3) soit solution.

Remarque : il faut exploiter au mieux cette proposition:
par exemple pour n=4 et $P_1=6,P_2=12,P_3=18,P_4=3$ alors $d_{12}=d_{13}=d_{23}=6$ et $d_{14}=d_{24}=d_{34}=3$ et ainsi les hypothèses de la proposition 1 ne sont pas remplies mais en revanche:
si on restreint le système à P_1,P_2 et P_3 avec n=3 alors les hypothèses de la proposition 1 deviennent valides et on sait alors exhiber rapidement toutes les $A_6^3=120$ solutions au problème associé(on les a toutes car les d_{ij} sont tous égaux), il est ensuite plus aisé de chercher les 4-uplets solutions en prolongeant les triplets solutions du sous problème il faut ainsi résoudre le système $\begin{cases}r_4\neq r_1 \bmod 3\\ r_4\neq r_2 \bmod 3\\ r_4\neq r_3 \bmod 3\end{cases}$ ce qui nous permet de dire que pour avoir des solutions il faut et il suffit que $r_1=r_2 \bmod 3$ ou $r_2=r_3 \bmod 3$ ou $r_1=r_3 \bmod 3$.Par exemple les quadruplets (1,4,2,3) et(1,2,5,3) sont solutions.

Proposition C.2: une condition suffisante pour la non-existence de solutions est que: $\mu = \underset{i<j}{PPCM}(d_{ij}) \le n-1$.

Preuve: Raisonnons par l'absurde:
$\exists(r_1,...,r_n)$ tel que $\forall(i,j), i \ne j : r_i \ne r_j \bmod(d_{ij})$ c'est à dire que d_{ij} ne divise pas r_i-r_j et donc, puisque μ est un multiple de d_{ij}, μ ne divise pas r_i-r_j ce qui s'écrit $r_i \ne r_j \bmod(\mu)$, ceci signifie que les classes de tous les r_k modulo μ, avec $k \in [\![1,n]\!]$, sont toutes deux à deux différentes et ainsi dans $\mathbf{Z}/\mu\mathbf{Z}$ on a n classes distinctes dans un espace à μ classes (en effet l'espace $\mathbf{Z}/p\mathbf{Z}$ avec p entier naturel possède p éléments)ce qui contredit l'hypothèse $\mu \le n-1$.

Exemple : pour n=7 avec P_1=3, P_2=6, P_3=9, P_4=12, P_5=15, P_6=21, P_7=33 il vient que d_{24}=6 et $d_{12}=d_{13}=d_{14}=d_{15}=d_{16}=d_{17}=d_{23}=d_{25}=d_{26}=d_{27}=d_{34}=d_{35}=d_{36}=d_{37}=d_{45}=d_{46}=d_{47}=d_{56}=d_{57}=d_{67}=3$ il apparaît que $\underset{i<j}{PPCM}\, d_{ij} \le 7-1$ donc d'après la proposition C.2 il n'existe pas de solutions.

Avec une idée similaire à celle de la proposition C.2 on construit une condition suffisante un peu plus fine:

Proposition C.3: si $\exists i_0 \in [\![1,n]\!], k \in [\![1,n]\!]$ et k termes $d_{i_0 j_1},..,d_{i_0 j_k}$ tels que $\mu = \underset{(m,l)\in A}{PPCM}(d_{i_0 j_1},..,d_{i_0 j_k},d_{j_m j_l}) \le k$ avec A={(m,l)$\in [\![1,k]\!]$/ m$\ne l$ } alors il n'existe pas de solutions au critère (5).

Preuve: Raisonnons par l'absurde : $\exists(r_1,...,r_n)$ tel que $\forall(i,j), i \ne j : r_i \ne r_j \bmod(d_{ij})$ alors nécessairement:

$$\begin{cases} r_{i_0} \ne r_{j_1} \bmod(d_{i_0 j_1}) \\ ... \\ r_{i_0} \ne r_{j_k} \bmod(d_{i_0 j_k}) \\ r_{j_m} \ne r_{j_l} \bmod(d_{j_l j_m}) \end{cases} \text{ et donc } \begin{cases} r_{i_0} \ne r_{j_1} \bmod \mu \\ ... \\ r_{i_0} \ne r_{j_k} \bmod \mu \\ r_{j_m} \ne r_{j_l} \bmod \mu \end{cases}$$

et ainsi dans $\mathbf{Z}/\mu\mathbf{Z}$ on a k classes $\overline{r_{j_1}},..,\overline{r_{j_k}}$ qui sont toutes distinctes de $\overline{r_{i_0}}$ d'après les k premières équations et elles sont toutes distinctes entre elles d'après les autres équations. On a Ainsi k+1classes distinctes ce qui contredit l'hypothèse $\mu \le k$ pour la même raison que dans la preuve précédente.

Proposition C.4: si le système ne vérifie pas la proposition C.1 c'est à dire si $\min_{i<j}(d_{ij}) < n$ mais qu'un seul des d_{ij} en question est plus petit que n alors il existe quand même des solutions au critère (5).

Préliminaire : On appellera sous-système issu du critère (5) formé par les "P_i" tel que i$\in$ J, J étant une partie de l'ensemble {1,..,n}, le système $\forall(i,j)\in J^2, i \ne j : r_i \ne r_j \bmod(d_{ij})$. Si on connaît toutes les solutions associées à un sous-système de notre système alors nécessairement toute solution au critère (5) est un prolongement d'une des solutions du sous-système. Plus généralement tout p-uplet extrait d'un n-uplet solution est solution du sous-système associé à ce p-uplet.

Preuve: Considérons une solution (r_1,..,r_n) au critère (5) alors pour toute partie J de {1,..,n} et donc à fortiori pour celle associée à notre sous-système, les "r_i" de notre solution vérifie:
$\forall(i,j)\in J^2, i \ne j : r_i \ne r_j \bmod(d_{ij})$.
Et donc ce n-uplet solution est bien un prolongement d'une des solutions du sous-système.

Preuve de C.4: notons $d_{i_0 j_0}$ le seul PGCD qui soit inférieur strictement à n, on regarde alors le sous-système S', formé par tous les "P_i" sauf P_{i_0}, pour ce système on note $\alpha' = \inf_{i\ne i_0, j\ne i_0} d_{ij} \ge n > n-1$ et donc, d'après la proposition C.1, on connaît des solutions pour ce sous-système. Pour terminer la preuve il suffit de trouver un r_{i_0} qui vérifie:

$$\begin{cases} r_{i_0} \ne r_{j_0} \bmod(d_{i_0 j_0}) \\ r_{i_0} \ne r_j \bmod(d_{i_0 j}); \forall j \ne i_0, j \ne j_0 \end{cases}$$ en utilisant le préliminaire.

On sait que pour les solutions au sous-problème S', on peut choisir les r_i comme on le souhaite entre 0 et $\alpha'-1$ (voir preuve de la proposition C.1),prenons par exemple r_{j_0} dans $[\![0, d_{i_0 j_0}-1]\!]$ et notons k_0 le plus grand entier naturel k tel que: $k.d_{i_0 j_0} + r_{j_0} \le n-1$,alors $k_0 \le (n-1-r_{j_0})/d_{i_0 j_0}$, et puisque les d_{ij} sont tous supérieurs à 2 il vient que $k_0 \le$ (n-1)/2,

et donc comme k_0 est en fait le nombre d'entiers de la forme $r_{j_0} + k.d_{i_0 j_0}$ avec k dans $[\![1, k_0]\!]$, on peut placer toutes ces valeurs dans la liste des r_i, avec $i \neq i_0$ et $i \neq j_0$ (il est possible que k_0=0 mais cela ne gène pas la démonstration car alors il n'existe pas d'entiers k dans $[\![1, k_0]\!]$).

Ainsi on peut choisir r_{i_0} dans $[\![0, n-1]\!]$ qui soit différent des autres r_i (il y a au moins un choix possible car il y a n-1 "r_i" avec $i \neq i_0$) et de plus on ne peut pas avoir $r_{i_0} = r_{j_0} \bmod(d_{i_0 j_0})$ car les places correspondantes possibles sont déjà occupées. On a ainsi construit un n-uplet solution

En fait on a plus d'une solution, en effet on peut choisir la place de r_{j_0} entre 0 et n-1, il suffit juste de s'assurer que tous les nombres qui lui sont égaux modulo $d_{i_0 j_0}$ et qui sont inférieurs ou égaux à n-1 sont des valeurs prises par d'autres r_i de manière à ce qu'ils ne puissent pas être pris par r_{i_0}.

Ceci nous donne un nombre de solutions au moins égal à $n(n-1-k_0)! A_{n-1}^{k_0}$, formule dans laquelle k_0 représente le nombre de valeurs $r_{j_0} + k.d_{i_0 j_0}$ avec k entier relatif tel que ces valeurs soient dans $[\![0, n-1]\!]$, n représente le nombre de possibilités pour le choix de r_{i_0}, $A_{n-1}^{k_0}$ le nombre de façons de placer les $k.d_{i_0 j_0} + r_{j_0}$ et $(n-1-k_0)!$ le nombre de façons de placer les autres valeurs.

Annexe D – Le problème dans le cas général

Dans cette annexe, nous allons démontrer la proposition suivante :

Proposition D : *Soient deux tâches T_i et T_j non préemptibles, s'exécutant dès leur activation, de durée d'exécution respective C_i et C_j, et soit p leur PGCD. Ces deux tâches ne se perturbent pas entre elles pendant leur exécution si et seulement si*

$$\begin{pmatrix} r_{j,1}-r_{i,1}\neq 0 \bmod(p) \\ r_{j,1}-r_{i,1}\neq 1 \bmod(p) \\ \dots \\ r_{j,1}-r_{i,1}\neq C_i-1 \bmod(p) \end{pmatrix} et \begin{pmatrix} r_{i,1}-r_{j,1}\neq 0 \bmod(p) \\ r_{i,1}-r_{j,1}\neq 1 \bmod(p) \\ \dots \\ r_{i,1}-r_{j,1}\neq C_j-1 \bmod(p) \end{pmatrix}$$

Pour cela, nous devons passer par quelques étapes intermédiaires :

Rappel : Deux tâches τi<ri,Ci=1,Di,Pi> et τj<rj,Cj=1,Dj,Pj> non préemptibles qui s'exécutent dès leur activation, se perturbent entre elles si et seulement si il existe un entier x tel que x≡ri[Pi] et x≡rj[Pj] .

Preuve : (Cf. Annexe A)

Généralisons ensuite cela à des tâches de durée d'exécution plus grande que 1.

Proposition D.1: Soient deux tâches τj<rj,Cj,Dj,Pj> et τi<ri,Ci,Di,Pi> de durée d'exécution quelconque, non préemptibles et qui s'exécutent dès leur activations. La tâche τj<rj,Cj,Dj,Pj> perturbe l'exécution de la tâche τi<ri,Ci,Di,Pi> si et seulement si il existe x entier naturel tel que { x≡ri[Pi] ou x≡(ri+1)[Pi] ou … ou x≡(ri+Ci−1)[Pi] } et x≡rj[Pj]. (i.e. τj est activée dans un intervalle du type [ri+kPi ; ri+kPi+Ci−1]).

Preuve : Revenons à la définition que nous pouvons donner à la propriété « une tâche τj<rj,Cj,Dj,Pj> perturbe l'exécution de la tâche τi<ri,Ci,Di,Pi> » : elle consiste à trouver (k,k') couple d'entiers naturels et c_i entier appartenant à l'intervalle $[\![0;C_i-1]\!]$ de sorte que : $r_i+kP_i+c_i=r_j+k'P_j$. Dans le sens droite gauche, le si et seulement si est alors évident. Dans le sens gauche droite, il suffit d'écrire que :

(Il existe (k,k') couple d'entiers naturels et c_i entier appartenant à l'intervalle $[\![0;C_i-1]\!]$ de sorte que : $r_i+kP_i+c_i=r_j+k'P_j$)

⇔ (Il existe c_i entier appartenant à l'intervalle $[\![0;C_i-1]\!]$ et il existe x entier naturel solution de $\begin{cases} x=r_i+kP_i+c_i \\ x=r_j+k'P_j \end{cases}$)

⇔ (Il existe c_i entier appartenant à l'intervalle $[\![0;C_i-1]\!]$ et il existe x entier naturel solution de $\begin{cases} x=r_i+c_i \bmod(P_i) \\ x=r_j \bmod(P_j) \end{cases}$)

⇔ (Il existe x entier naturel solution d'un des systèmes : $\begin{cases} x=r_i+0 \bmod(P_i) \\ x=r_j \bmod(P_j) \end{cases}$, $\begin{cases} x=r_i+1 \bmod(P_i) \\ x=r_j \bmod(P_j) \end{cases}$, … , $\begin{cases} x=r_i+C_i-1 \bmod(P_i) \\ x=r_j \bmod(P_j) \end{cases}$)

Proposition D.2 : Soient deux tâches τj<rj,Cj,Dj,Pj> et τi<ri,Ci,Di,Pi> de durée d'exécution quelconque, non préemptibles et qui s'exécutent dès leur activations.

La tâche τj<rj,Cj,Dj,Pj> ne perturbe pas l'exécution de la tâche τi<ri,Ci,Di,Pi> si et seulement si

$$\begin{pmatrix} r_{j,1} - r_{i,1} \neq 0 \bmod(p) \\ r_{j,1} - r_{i,1} \neq 1 \bmod(p) \\ \ldots \\ r_{j,1} - r_{i,1} \neq C_i - 1 \bmod(p) \end{pmatrix}$$

Preuve : Tout d'abord, faisons un rappel sur le théorème chinois généralisé (TRCG) [16] : Soient p1, p2,…,pm des entiers positifs, et r1, r2, …, rm des entiers. Il existe exactement un entier x, tel que pour tout entier r, r≤x<r+ppcm(pi)i=1..m, x≡rj [pj] 1≤j≤m si et seulement si ri≡rj [pgcd(pi,pj)], 1≤i<j≤m.

L'application de ce théorème sur le résultat de la proposition D.1 donne la formule équivalente : { ri≡rj[pgcd(Pi,Pj)] ou ri+1≡rj[pgcd(Pi,Pj)] ou … ou ri+Ci−1≡rj[pgcd(Pi,Pj)] }.

τj ne perturbe pas τi si et seulement si {(ri≠rj)[pgcd(Pi,Pj)] et (ri+1≠rj)[pgcd(Pi,Pj)] et… et ri+Ci−1≠rj[pgcd(Pi,Pj)]}. Donc on peut écrire de façon équivalente {(rj−ri≠0)[pgcd(Pi,Pj)] et (rj−ri≠1)[pgcd(Pi,Pj)] et… et rj−ri≠Ci−1[pgcd(Pi,Pj)], soit donc le résultat à démontrer. Plaçons sur la figure D.1 suivante les valeurs interdites de rj−ri dans $\mathbb{Z}/p\mathbb{Z}$, avec p=pgcd(Pi,Pj), et observons les valeurs restantes.

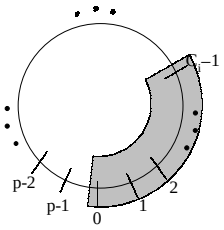

Figure D.1 Valeurs interdites de rj−ri dans $\mathbb{Z}/p\mathbb{Z}$ avec p=pgcd(Pi,Pj)

Sur cette figure, les valeurs interdites pour rj−ri sont grisées, les valeurs possibles sont le complémentaires dans $\mathbb{Z}/p\mathbb{Z}$ de l'ensemble des valeurs grisées. On peut remarquer qu'il est nécessaire et suffisant que Ci<pgcd(Pi,Pj) pour que cet ensemble soit non vide, car dans le cas contraire, toutes les valeurs de $\mathbb{Z}/p\mathbb{Z}$ seraient grisées. Les valeurs autorisées donc l'ensemble que l'on peut caractériser dans $\mathbb{Z}/p\mathbb{Z}$ par (rj−ri)≥Ci[pgcd(Pi,Pj)].

Proposition D.3 : Soient deux tâches τj<rj,Cj,Dj,Pj> et τi<ri,Ci,Di,Pi> de durée d'exécution quelconque, non préemptibles et qui s'exécutent dès leur activations.

Ces deux tâches τj<rj,Cj,Dj,Pj> et τi<ri,Ci,Di,Pi> ne se perturbent pas entre elles pendant leur exécution si et seulement

$$\text{si} \begin{pmatrix} r_{j,1} - r_{i,1} \neq 0 \bmod(p) \\ r_{j,1} - r_{i,1} \neq 1 \bmod(p) \\ \ldots \\ r_{j,1} - r_{i,1} \neq C_i - 1 \bmod(p) \end{pmatrix} \text{et} \begin{pmatrix} r_{i,1} - r_{j,1} \neq 0 \bmod(p) \\ r_{i,1} - r_{j,1} \neq 1 \bmod(p) \\ \ldots \\ r_{i,1} - r_{j,1} \neq C_j - 1 \bmod(p) \end{pmatrix}$$

Preuve : C'est immédiat avec la proposition D.2 : τj<rj,Cj,Dj,Pj> et τi<ri,Ci,Di,Pi> ont un rôle symétrique.

Cette proposition est donc un outil permettant de choisir la date de réveil des tâches régulières de façon à ce qu'elles ne soient jamais concurrentes. Cela limite donc le choix de la différence (rj−rj)[pgcd(Pi,Pj)] à la partie non grisée de la figure D.2.

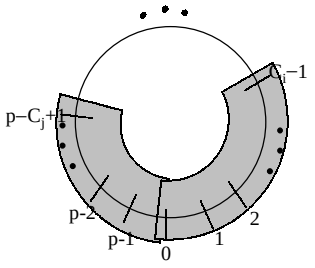

Figure D.2 - Les valeurs possibles pour le choix de (rj−ri) pour éviter leur concurrence, avec p=pgcd(Pi,Pj)

Annexe E – D'autres résultats

Pour simplifier les notations nous utilisons dans cette annexe indifféremment le n-uplet $(r_1,\ldots,r_n)$ et le n-uplet $(r_{1,1},r_{2,1},\ldots,r_{n,1})$.

Proposition *Avant toute recherche des $r_{i,1}$, il faut là encore vérifier que tous les P_i sont non premiers entre eux deux à deux.*

Cette proposition découle en fait du résultat de l'annexe B : le problème initial consiste à trouver un couple d'entiers naturels (r_1,r_2) de sorte qu'il n'existe aucune solution en x dans l'ensemble des entiers naturels à des systèmes du type:

$S(1,2) \Leftrightarrow \begin{cases} x = r_1 + c_1 \bmod(P_1) \\ x = r_2 \bmod(P_2) \end{cases}$ avec c_1 entier naturel contenu dans l'intervalle : $[\![0; C_i - 1]\!]$. L'annexe B montre alors que les seuls cas qui répondent au problème posé sont les cas où $P_1 \wedge P_2 \neq 1$ et les solutions sont les couples (r_1,r_2) tels que $r_1 + c_1 \neq r_2 \bmod(d)$.

Rappelons ensuite le problème posé généralisé : soient deux tâches T_i et T_j non préemptibles, s'exécutant dès leur activation, de durée d'exécution respective C_i et C_j, et soit p leur PGCD. On cherche l'ensemble des $r_{i,1}$ vérifiant

$$\begin{pmatrix} r_{j,1} - r_{i,1} \neq 0 \bmod(p) \\ r_{j,1} - r_{i,1} \neq 1 \bmod(p) \\ \ldots \\ r_{j,1} - r_{i,1} \neq C_i - 1 \bmod(p) \end{pmatrix} et \begin{pmatrix} r_{i,1} - r_{j,1} \neq 0 \bmod(p) \\ r_{i,1} - r_{j,1} \neq 1 \bmod(p) \\ \ldots \\ r_{i,1} - r_{j,1} \neq C_j - 1 \bmod(p) \end{pmatrix} \quad \textbf{(7)}$$

On dispose pour ce problème de quelques outils analytiques pour caractériser des solutions. Pour l'instant reformulons le problème (7).

Critère E.1 : le problème (7) se réduit à trouver l'ensemble des r_i de sorte que :
$\forall (i,j) \in [\![1,n]\!]^2, i \neq j, \forall \rho \in [\![-(C_i - 1); C_j - 1]\!] : r_i - r_j \neq \rho \bmod(d_{ij})$

Proposition E.1 Une condition nécessaire à l'existence de solutions à ce critère E.1 est que $\forall (i,j) \in [\![1,n]\!]^2, i \neq j : (C_i - 1) + (C_j - 1) < d_{ij}$, où d_{ij} représente le pgcd de P_i et de P_j.

Preuve: D'après le critère E.1, on a $r_i - r_j \neq \rho \bmod(d_{ij})$ ainsi pour pouvoir trouver une solution il faut donc que, modulo d_{ij}, les ρ occupent au plus d_{ij} classes distinctes or ils se succèdent et puisqu'ils sont au nombre de $(C_i - 1) + (C_j - 1) + 1$, ils occupent donc k classes distinctes où k est le plus petit nombre entre d_{ij} et $(C_i - 1) + (C_j - 1) + 1$ et ainsi il faut que $(C_i - 1) + (C_j - 1) + 1 \leq d_{ij}$ et ce pour tout couple (i,j), soit $(C_i - 1) + (C_j - 1) < d_{ij}$, pour tout couple (i,j).

Proposition E.2: Une condition nécessaire sur les r_k avec k variant entre 1 et n pour qu'ils forment un n-uplet solution du critère E.1 est : $\forall (i,j) \in [\![1,n]\!]^2, i \neq j : |r_i - r_j| > \sup(C_i - 1, C_j - 1)$.

Preuve: supposons que le n-uplet $(r_1,..,r_n)$ soit solution alors pour tout i et pour tout j compris entre 1 et n et différents l'un de l'autre: $r_i - r_j \neq \rho \bmod(d_{ij})$ et à fortiori $r_i - r_j \neq \rho$ ceci avec ρ qui varie entre $-(C_i - 1)$ et $(C_j + 1)$ ce qui montre le résultat.

Pour toute la suite de l'annexe on note $\beta = \sup_p C_p$.

Proposition E.3: si $\alpha = \inf_{i<j}(d_{ij}) \geq n\beta$ alors il existe au minimum n! solutions au problème posé.
(n! est un minimum car la démonstration exhibe n! solutions mais ne nous dit pas que celles-ci sont les seules.)

Preuve: prenons le n-uplet $r_1=0, r_2=\beta, .., r_n=(n-1)\beta$ et vérifions qu'il est bien solution: il faut vérifier pour cela que tous les $r_k + \rho$ sont différents des r_i modulo d_{ik} pour $\begin{cases} i \in [\![1,n]\!] \\ k \in [\![2,n]\!], k \neq i \end{cases}$.

Les r_k, avec k variant entre 1 et n, sont tous deux à deux distants d'au moins β donc puisque ρ est compris entre -$\beta+1$ et $\beta-1$:les $r_k+\rho$ sont tous distincts entre eux et ils sont tous distincts de r_1, et ce $\begin{cases}\forall i\in[\![1,..,n]\!]\\ \forall k\in[\![2,..,n]\!], k\neq i\end{cases}$, de plus ces $r_k+\rho$ sont tous compris entre 0 et α-1 or les classes modulo α de tous les nombres entre 0 et α-1 sont toutes distinctes, on voit ainsi que le n-uplet considéré vérifie le critère E.1.
De plus, comme pour la proposition C.1, toutes les permutations de cette solution sont encore solutions du problème posé et pour $k\beta\leq\alpha<(k+1)\beta$ on peut exhiber A_k^n solutions (dans le cas de la première partie β =0 et donc $\alpha=k$ et ce nombre de solutions A_k^n est l'analogue du A_α^n de la proposition C.1).

Prenons par exemple:n=4 et $P_1=P_2=P_3=16,P_4=800$ et $C_1=3,C_2=C_3=2,C_4=4$ alors le système
associé vérifie la proposition E.1 et ainsi on regarde s'il vérifie la proposition E.3: $\beta=4,\alpha=16$ et donc puisque $16\geq 4\times 4$ on sait exhiber 24 solutions.

Remarque E.1: la condition de la proposition E.3 n'est pas nécessaire à l'existence de solutions:
reprenons un exemple précédemment utilisé dans la première partie: n=4 et $P_1=84,P_2=6,P_3=21,P_4=12$, mais maintenant $C_2=C_3=C_4=1$ et $C_1=2$ il vient:$d_{12}=6,d_{13}=21,d_{14}=12,d_{23}=3,d_{24}=6$ et $d_{34}=3$ et alors on vérifie la proposition E.1 mais pas la condition de la proposition E.3 et pourtant (0,2,3,4) est solution.

Rapports de recherche LISI Année 2000 :

00 001 : GENIET D., Validation d'Applications Temps-Réel à Contraintes Strictes à l'Aide de Langages Rationnels, janvier 2000.

00 002 : GENIET D., DUBERNARD J.-P., Association de Langages Rationnels et de Fonctions Génératrices pour l'Ordonnancement de Tâches Apériodiques dans les Systèmes Temps-Réel Distribués à Contraintes Strictes, février 2000.

00 003 : SARDET E., Une Approche pour la Représentation des Catalogues de Composants Industriels : le modèle P-Lib, avril 2000.

00 004 : DAVID L., COTTET F., GROLLEAU E., Maîtrise de la Gigue Temporelle avec les Algorithmes d'Ordonnancement DM et ED, avril 2000.

Résumé :
L'utilisation d'un ordonnancement en ligne selon les algorithmes classiques (RM, DM et ED) ne permet pas au cours de l'exécution d'une application temps réel, de maîtriser le phénomène de gigue temporelle de certaines tâches. Cette gigue s'avère d'autant plus néfaste lorsque les tâches ont une période qui doit être strictement respectée (acquisition de données, commande d'un actionneur, etc.). Nous nous proposons donc dans ce travail de fournir une méthode permettant d'annuler la gigue temporelle pour des tâches dont les contraintes de régularité sont strictes, ceci dans le cadre des ordonnancements basés sur l'échéance (DM, ED).

Mots-Clés :
Gigue temporelle, ordonnancement en ligne, algorithmes d'ordonnancement.

Abstract:
One typical feature of real-time systems is concurrent processing with timing constraints. These timing constraints, for example, can deal not only with the deadline meeting but also with the task execution regularity. For application in hard real-time systems, repetitive requests are usually designed with the periodic task model from Liu and Layland. However, in this model, the on-line scheduling classical algorithms do not allow task execution regularity control. Yet, unpredictable delays or jitter in the task periodic executions may be quite disturbing for some real-time applications such as process controls (sampled data systems, P.I.D. control, etc.). We propose in the report a method based on the modification of the task temporal parameters in the well known DM and EDF algorithm context, in order to master the jitter in particular conditions.

Keywords:
Temporal jitter, on line scheduling, scheduling algorithms.

Rapports disponibles auprès de :

M. Yamine AIT-AMEUR
LISI - ENSMA
Site du Futuroscope – BP 40109 – 86961 FUTUROSCOPE Cedex – FRANCE
Tél : +33 (0) 5 49 49 80 63 – Fax : +33 (0) 5 49 49 80 64
Web : http://www.lisi.ensma.fr