Burkina Faso
Année accadémique 2006-2007
Université Polytechnique de Bobo-Dioulasso
Ecole Supérieure d'Informatique

Mémoire de DEA en Informatique

Sujet :

Extension du contexte de mise en œuvre des ordonnancements P-équitables

Moustapha BIKIENGA

Encadrants :

Annie CHOQUET-GENIET, Université de Poitiers
Mesmin DANDJINOU, Université Polytechnique de Bobo-Dioulasso

Table des matières

Dédicace

Je dédie ce mémoire à mon père Feu Issa BIKIENGA.

Remerciements

J'adresse mes vifs remerciements :
- à mes encadrants le Professeur Annie CHOQUET-GENIET et le Docteur Mesmin DANDJINOU pour leurs disponibiltés et leurs conseils ;
- au responsable du DEA en Informatique le Professeur Théodore TAPSOBA ;
- à MALO Sadouanouan pour son aide et sa gentillesse ;
- aux enseignants et aux personnels de l'Ecole Supérieure d'Informatique ;
- à ma famille pour son soutien indéfectible tout au long de mon cursus scolaire et universitaire .

Introduction

De nos jours la grande majorité des procédés (allumage de véhicule, pilotage de robot ou d'avion, transmission multimédia, ...) sont contrôlés par des applications informatiques. La particularité de ces applications est qu'elles "ont conscience" de l'écoulement du temps et qu'elles prennent leurs décisions en fonction de celui-ci. On parle alors de systèmes temps-réel. D'après [8] :
"Un système temps réel est un systéme dans lequel l'exactitude des applications ne dépend pas seulement de l'exactitude du résultat, mais aussi du temps auquel ce résultat est produit. Si les contraintes temporelles de l'application ne sont pas respectées, on parle de défaillance du système. Il est donc essentiel de pouvoir garantir le respect des contraintes temporelles du système. Ceci nécessite que le système permette un taux d'utilisation élevé, tout en respectant les contraintes temporelles."
La définition implique que les sytèmes temps-réel soient soumis à deux types de contraintes : contraintes fonctionnelles et contraintes temporelles. La réalisation d'applications temps-réel nécessite donc aussi bien une validation fonctionnelle qu'une validation temporelle. Nous nous intéressons dans notre travail à la validation temporelle.
La taille grandissante des applications conduit de plus en plus à une utilisation de calculateurs ayant de grandes puissances de calcul. Selon [1] pour une puissance de calcul équivalente, les calculateurs munis de plusieurs processeurs ont un coût moins élevé que ceux munis d'un unique processeur très performant. Dans notre cas, nous nous intéresserons aux algorithmes P-équitables dans un contexte multiprocesseurs. Nous chercherons plus précisement à étendre les résultats de [7].
La suite du document s'articulera autour de deux grandes parties. La première fera une présentation de l'état de l'art et la deuxième présentera les résultats du stage.

Première partie

État de l'art

Chapitre 1

Applications temps réel

Dans ce chapitre nous nous sommes appuyés sur [1] pour donner des défintions d'ordre général relatives aux applications temps réel.

1.1 Système temps réel

1.1.1 Modèle de système considéré

Nous utilisons, dans notre rapport, comme formalisme de modélisation des systèmes, celui qui est le plus utilisé dans la communauté de l'ordonnancement temps réel. Dans ce formalisme, le temps est une grandeur physique mesurable qui permet de caractériser les traitements à effectuer. Le système se compose de demandeurs de ressources, de serveurs de ressources et d'un composant central(voir figure 1.1).

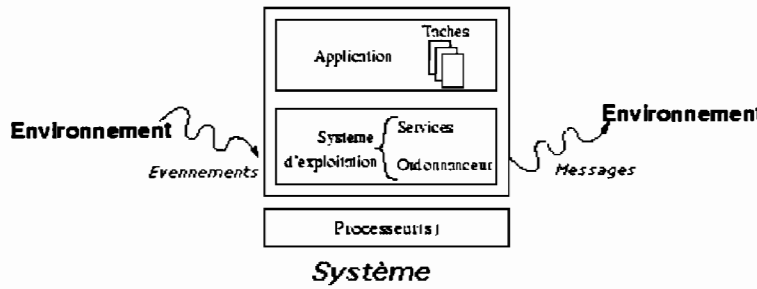

FIG. 1.1 – Modèle de système temps réel

Le composant central est le support d'exécution. Il est chargé de répartir les demandes en ressources de l'application entre les ressources matérielles disponibles. Le système repose sur un ou plusieurs processeurs pour effectuer les traitements.

1.1.2 Tâche

Une tâche est une séquence d'opérations du processeur qui fournit un service. Ce service peut être rendu plusieurs fois. Chacune des exécutions est appelée instance de tâche. L'instance est assimilable au travail. Avant de définir plus formellement le travail, donnons quelques notations qui seront utilisées tout au long du rapport.

Notation : Soit a et b deux entiers naturels avec $a < b$

- $[a, b)$ désigne l'ensemble fini des entiers compris entre a et b, a inclus et b exclus.

$$[a, b) = \{a, \ldots, b-1\}$$

-

$$[a, b] = [a, b+1) = \{a, \ldots, b\}$$

-

$$(a, b] = [a+1, b+1) = \{a+1, \ldots, b\}$$

Définition 1 (Travail) *Un travail j sera caractérisé par le triplet (a, e, d). Un instant d'arrivée a, un temps d'exécution e et une échéance absolue d(voir figure 1.2). Le travail j doit recevoir e unités d'exécution dans l'intervalle $[a, d)$. Une instance temps réel est une collection (finie ou infinie) de travaux : $J = \{j_1, j_2, \ldots\}$*

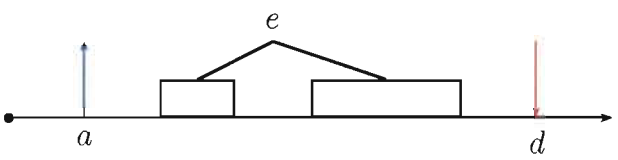

FIG. 1.2 – Travail $j = (a, e, d)$

En fonction de la répartition dans le temps des dates d'arrivées des travaux qui se répètent, on distingue les tâches apériodiques, périodiques et sporadiques.

Définition 2 (Tâche périodique) *Une tâche périodique τ_i est caractérisée par le quadruplet (r_i, C_i, D_i, T_i) (voir figure 1.3) :*

- *r_i est la date d'arrivée de la première instance de la tâche τ_i ;*
- *C_i est la pire durée d'exécution, il spécifie une limite supérieure sur le temps d'exécution de chaque instance de la tâche τ_i ;*
- *D_i est l'échéance relative, elle dénote la durée séparant l'arrivée d'une instance et son échéance ;*
- *T_i est la période, c'est l'intervalle de temps qui sépare l'arrivée de deux instances successives de τ_i.*

Le facteur de charge d'une tâche est défini par

$$U(\tau_i) = \frac{C_i}{T_i}$$

Nous parlerons de systèmes périodiques :
- à échéances sur requête si toutes les tâches τ_i du système ont une échéance égale à la période : $D_i = T_i$;
- à échéances contraintes si toutes les tâches τ_i du système ont une échéance inférieure à la période : $D_i \leq T_i$;
- à échéances arbitraires, s'il n'y a pas de contrainte entre l'échéance et la période ;
- à départs simultanés, si toutes les tâches démarrent au même instant : $\forall \tau_i$ $r_i = 0$;
- à départs différés si les dates d'arrivée sont quelconques.

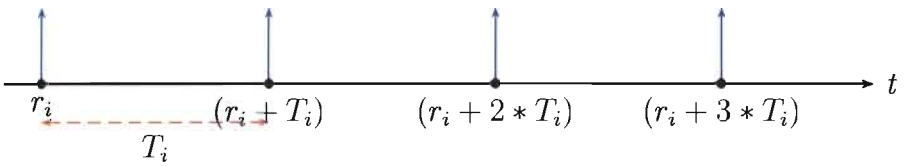

FIG. 1.3 – Tâche périodique de date d'arrivée r_i et de période T_i

Définition 3 (Tâche sporadique) *Une tâche sporadique τ_i est caractérisée par le triplet (C_i, D_i, T_i)(voir figure 1.4), avec :*
- *C_i qui est la pire durée d'exécution, il spécifie une limite supérieure sur le temps d'exécution de chaque instance de la tâche τ_i ;*
- *D_i qui est l'échéance relative, elle dénote la durée séparant l'arrivée d'une instance et son échéance ;*
- *T_i qui est la période, c'est-à-dire l'intervalle de temps **minimal** qui sépare l'arrivée de deux instances successives de τ_i.*

Définition 4 (Tâche apériodique) *Une tâche apériodique τ_i est caractérisée par le couple (C_i, D_i). Les instances de la tâche peuvent être créées à tout instant et les définition de C_i et D_i sont les mêmes pour une tâche sporadique.*

Définition 5 *Soit $P = \{P_1, \ldots, P_m\}$ une plate-forme de m processeurs.Nous distinguons trois types de plates-formes présentée ci-après.*

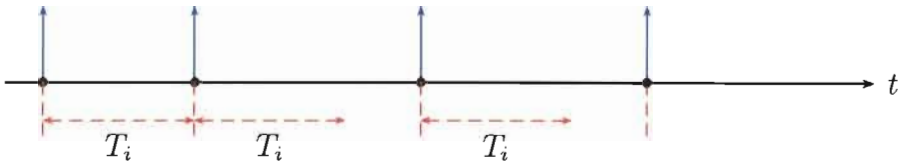

FIG. 1.4 – Tâche sporadique de période T_i

- *Les plates-formes à processeurs identiques : les processeurs sont interchangeables et ont la même puissance de calcul ;*
- *Les plates-formes à processeurs uniformes : chaque processeur d'une plate-forme uniforme est caractérisé par sa capacité de calcul. Par exemple si un travail j_i s'exécute sur le processeur $P_k$de capacité s_k pendant t unités de temps alors j_i a réalisé $s_k \times t$ unités de travail ;*
- *Les plates-formes à processeurs indépendants : soit $J = \{j_1, j_2, \ldots\}$ une instance temps réel et $P = \{P_1, \ldots, P_m\}$ une plate-forme de m processeurs indépendants. A chaque couple (j_i, P_k) est associé le taux d'exécution $s_{i,k}$ de sorte que lorsque le travail j_i s'exécute sur le processeur P_k pendant t unités de temps alors il a réalisé $(s_{i,j} \times t)$ unités de travail.*

Nous supposons dans le cadre de notre stage, ne travailler qu'avec des plates-formes à processeurs identiques.

1.2 Ordonnancement temps réel

Définition 6 *L'ordonnancement d'un système de tâche τ est une fonction S définie par*

$$S : \tau \times N \rightarrow \{0, 1\}$$

$$S(\tau_i, t) = \begin{cases} 1 & si\ \tau_i\ est\ ordonnancée \\ 0 & sinon \end{cases}$$

En d'autres termes, l'ordonnancement consiste à déterminer une séquence infinie du type **(date, identifiant_tâche)** pour chaque processeur. Chaque élément correspond à l'élection d'une tâche à exécuter.

L'ordonnancement d'un système peut être décrit en utilisant les qualificatifs que nous listons ci-après.

- **Multiprocesseur/monoprocesseur :** l'ordonnancement est de type monoprocesseur si toutes les tâches ne peuvent s'exécuter que sur un seul et même processeur. Si plusieurs processeurs sont disponibles dans le système, l'ordonnancement est multiprocesseur.

- **Stratégie globale/Stratégie par partitionnement :** dans le cas où plusieurs processeurs sont utilisés, on dira que la stratégie est globale si elle s'applique sur l'entièreté de la plate-forme multiprocesseur. À l'opposé, si le système de tâches est partionné en sous-ensembles et ordonnancé sur des processeurs dédiés, on parle de stratégie par partitionnnement. Dans une stratégie globale il y a une seule file d'attente pour l'ensemble des processeurs alors que dans une stratégie par partitionnement il y a une file d'attente par processeur. Le principal problème dans le cas partitionné est la détermination des différentes files d'attentes.
- **Préemptif/non préemptif** l'ordonnancement est préemptif lorsque les préemptions (suspension de l'exécution d'une tâche) sont autorisées. Dans le cas contraire on parle d'ordonnancement non-préemptif.
- **En ligne/hors-ligne** un ordonnancement hors-ligne signifie que la séquence d'ordonnancement est prédéterminée à l'avance : dates de début d'exécution des tâches, de préemption/reprise éventuelle. Un ordonnancement en-ligne correspond au déroulement d'un algorithme qui tient compte des tâches présentes dans la file d'ordonnancement lors de chaque décision d'ordonnancement.
- **Statique/dynamique** les ordonnanceurs statiques fondent leurs décisions d'ordonnancement sur des paramètres assignés aux tâches du système avant leur activation. Les ordonnanceurs dynamiques fondent leurs décisions sur des paramètres qui varient en cours de fonctionnement du système.

Chapitre 2

Algorithmes P-équitables

Nous utilisons [7] et [2] pour présenter les algorithmes P-équitables.

2.1 Principe

Le principe général des algorithmes P-équitables est de répartir dans le temps l'ordonnancement des tâches proportionnellement à leurs besoins. Le coefficient de proportionnalité utilisé est le facteur de charge. Dans un ordonnancement idéalement équitable, chaque tâche τ_i reçoit exactement $U(\tau_i) * t$ unités de temps processeurs dans l'intervalle $[0, t)$. Comme l'ordonnancement est une fonction discrète, cet ordonnancement idéal ne peut être construit. La différence entre l'ordonnancement idéal et l'ordonnancement construit est le retard.

2.1.1 Retard

Soit une tâche τ_i à échéance sur requête avec comme date d'arrivée $r_i = 0$. Le retard de la tâche τ_i à la date t est :

$$retard(\tau_i, t) = U(\tau_i).t - \sum_{k=0}^{t-1} S(\tau_i, k)$$

Un ordonnancement est P-équitable si et seulement si

$$\forall(\tau_i, t) \in (\tau, N), -1 < retard(\tau_i, t) < +1$$

Cette condition implique qu'à t la tâche τ_i a reçu $\lfloor U(\tau_i).t \rfloor$ ou $\lceil U(\tau_i).t \rceil$ unités de temps d'allocation processeur. Autrement dit, chaque exécution de la tâche se fait dans un intervalle de temps prédéterminé. La notion de sous-tâche permet de formaliser ces contraintes d'exécution des tâches.

Définition 7 (Sous-tâche) *Toute tâche périodique τ_i peut être divisée en sous-tâches ayant une charge unitaire. Chaque sous-tâche τ_i^k doit se réaliser dans un intervalle de temps I_i^k appelé fenêtre de réalisation.*

$$I_i^k = [r_i^k, d_i^k) \text{ avec } \begin{cases} r_i^k = \lfloor \frac{k-1}{U(\tau_i)} \rfloor \\ d_i^k = \lceil \frac{k}{U(\tau_i)} \rceil \end{cases}$$

- *r_i^k est la date de "pseudo-activation" de τ_i^k*
- *d_i^k est la "pseudo-échéance" de τ_i^k*

Le "bit successeur" b_i^k de la sous-tâche τ_i^k est la différence entre les dates de pseudo-activation de τ_i^{k+1} et de pseudo-échéance de τ_i^k.

$$b_i^k = d_i^k - r_i^{k+1} = \lceil \frac{k}{U(\tau_i)} \rceil - \lfloor \frac{k}{U(\tau_i)} \rfloor$$

Exemple : Soit la tâche périodique $\tau_1 = (0, 3, 5, 5)$

Sous-tâche	Pseudo-activation	Pseudo-échéance	Fenêtre	Bit successeur
τ_1^1	0	2	[0,2)	1
τ_1^2	1	4	[1,4)	1
τ_1^3	3	5	[3,5)	0
τ_1^4	5	7	[5,7)	1
τ_1^5	6	9	[6,9)	1
τ_1^6	8	10	[8,10)	0

TAB. 2.1 – Sous-tâches de la tâche périodique $\tau_1 = (0, 3, 5, 5)$

Les fenêtres d'exécution des sous-tâches de τ_1 sont représentées dans la figure 2.1.

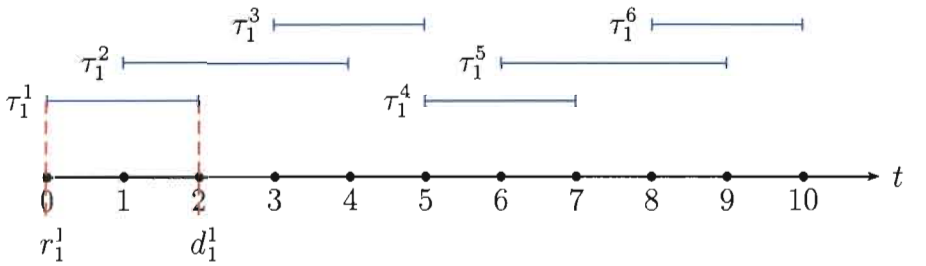

FIG. 2.1 – Sous-tâches de la tâche périodique $\tau_1 = (0, 3, 5, 5)$

2.1.2 Chaînes caractéristiques

En plus du retard, les algorithmes P-équitables utilisent les chaînes caractéristiques qui permettent de comparer les tâches. La chaîne caractéristique (traduction de l'expression anglaise "characteristic string") d'une tâche τ_i noté $\alpha(\tau_i)$ est une suite infinie des caractères $\{-, 0, +\}$.
La sous-chaîne caractéristique de τ_i à la date t est :

$$\alpha(\tau_i, t) = \alpha_{t+1}(\tau_i)\alpha_{t+2}(\tau_i)\ldots\alpha_{t'}(\tau_i) \text{ où } t' = mini : i > t : \alpha_i(\tau_i) = 0$$

avec

$$\alpha_t(\tau_i) = sign(U(\tau_i).(t+1) - \lfloor U(\tau_i).t \rfloor - 1)$$

2.2 Description des algorithmes

2.2.1 Description générale

Nous dirons à une date t qu'une tâche τ_i est :
- **en avance** si $retard(\tau_i, t) < 0$
- **en retard** si $retard(\tau_i, t) > 0$
- **ponctuelle** si $retard(\tau_i, t) = 0$

Dans le but de respecter les contraintes d'ordonnancement P-équitable, les tâches sont réparties en sous-ensembles. Ainsi une tâche est :
- **urgente** si elle est en retard et que $\alpha_t(\tau_i) \neq -$;
- **interdite** (tnegru) si elle est en avance et que $\alpha_t(\tau_i) \neq +$;
- **possible** (contending) si elle n'est ni urgente ni interdite.

Pour un système de tâches périodiques à départ simultané et à échéance sur requête $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ et une plateforme de m processeurs, les algorithmes P-équitables vont :

1. déterminer les sous-ensembles de tâches urgentes, possibles et interdites ;
2. allouer les tâches urgentes aux processeurs ;
3. allouer aux processeurs libres les tâches possibles les plus prioritaires.

Trois algorithmes P-équitables ont été proposés. Il s'agit de PF[7], PD[9] et PD^2[5]. Ils ne diffèrent que par la façon de trier les tâches possibles. Avant de présenter ces trois algorithmes, nous allons faire un rapprochement entre les notions de fenêtres d'execution et de chaînes caractéristiques.
Soit $\tau_i = (r_i, C_i, D_i, T_i)$ une tâche périodique telle que $r_i = 0$ et $D_i = T_i$. A t τ_i a reçue au minimum $\lfloor U(\tau_i).t \rfloor$ unités d'allocation processeur.
Notons $k_{min} = \lfloor U(\tau_i).t \rfloor + 1$ et $I_{min} = [r_i^{k_{min}}, d_i^{k_{min}})$ la fenêtre minimale.

Proposition 1 *Si* $[t, t+1)$ *n'est pas le dernier slot de la fenêtre minimale alors* $\alpha_t(\tau_i) = -$.

Rappel : Pour tout $x \in \mathcal{R}$, nous avons les propriétés suivantes :

$$\begin{cases} \lfloor x \rfloor & \leq & x & < & \lfloor x \rfloor + 1 \\ \lceil x \rceil - 1 & < & x & \leq & \lceil x \rceil \end{cases}$$

$$\alpha_t(\tau_i) = sign(U(\tau_i).(t+1) - \lfloor U(\tau_i).t \rfloor - 1) = sign(U(\tau_i).(t+1) - k_{min})$$

Preuve :
Si $[t, t+1)$ n'est pas le dernier slot de la fenêtre minimale alors
$t + 1 < d_i^{k_{min}}$
$t + 1 \leq d_i^{k_{min}} - 1$
$t + 1 \leq \lceil \frac{k_{min}}{U(\tau_i)} \rceil - 1 < \frac{k_{min}}{U(\tau_i)}$
$U(\tau_i)(t+1) < k_{min}$
$U(\tau_i)(t+1) - k_{min} < 0$
$\alpha_t(\tau_i) = -$

Proposition 2 *Si* $[t, t+1)$ *est le dernier slot de la fenêtre minimale alors* $\alpha_t(\tau_i) \in \{+, 0\}$.

Preuve :
Si $[t, t+1)$ est le dernier slot de la fenêtre minimale alors
$t + 1 = d_i^{k_{min}}$
$t + 1 = \lceil \frac{k_{min}}{U(\tau_i)} \rceil$
$t + 1 \geq \frac{k_{min}}{U(\tau_i)}$
$U(\tau_i)(t+1) \geq k_{min}$
$U(\tau_i)(t+1) - k_{min} \geq 0$
$\alpha_t(\tau_i) \in \{+, 0\}$

Proposition 3 *Si* $[t, t+1)$ *est le dernier slot de la fenêtre minimale et* $r_i^{k_{min}+1} \geq t + 1$ *alors* $\alpha_t(\tau_i) = 0$.

Preuve :
$r_i^{k_{min}+1} \geq t + 1$
$t + 1 \leq \lfloor \frac{k_{min}}{U(\tau_i)} \rfloor$

$t+1 \leq \dfrac{k_{min}}{U(\tau_i)}$
$U(\tau_i)(t+1) \leq k_{min}$
$U(\tau_i)(t+1) - k_{min} \leq 0$
$\alpha_t(\tau_i) \in \{0, -\}$
Par suite nous avons

$$\left.\begin{array}{l} \alpha_t(\tau_i) \in \{0, +\} \\ \alpha_t(\tau_i) \in \{0, -\} \end{array}\right\} \Rightarrow \alpha_t(\tau_i) = 0.$$

Exemple : Nous reprenons l'exemple de la tâche $\tau_1 = (0, 3, 5, 5)$. La figure 2.2 permet d'illustrer le rapprochement des notions de sous-tâches et de chaînes caractéristiques.

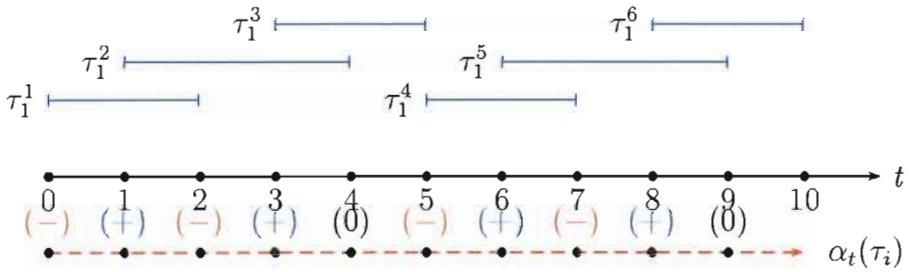

FIG. 2.2 – Sous-tâches et chaînes caractéristiques de la tâche périodique $\tau_1 = (0, 3, 5, 5)$

2.2.2 Description détaillée

Les descriptions que nous ferons des trois algorithmes P-équitable utilisent [2]. Ces descriptions se basent sur le fait que comparer deux tâches en utilisant les chaînes caractéristiques revient à comparer les sous-tâches.

Définition 8 (Ordre de priorité des tâches) *Pour deux tâches τ_i et τ_j notons $\succ$ la relation dont le sens est :*

$$(\tau_i \succ \tau_j) \Rightarrow \tau_i \text{ a une priorité strictement supérieure à } \tau_j$$

PF

Pour comparer deux tâches, PF compare les deux sous-tâches en-cours. Soient deux tâches τ_i et τ_j dont les sous-tâches en cours d'exécution sont τ_i^k et τ_j^p. On dira que τ_i est plus prioritaire que τ_j c'est-à-dire $\tau_i^k \succ \tau_j^p$ si :

Exemple : Considérons la tâche périodique $\tau_1 = (0, 6, 8, 8)$.

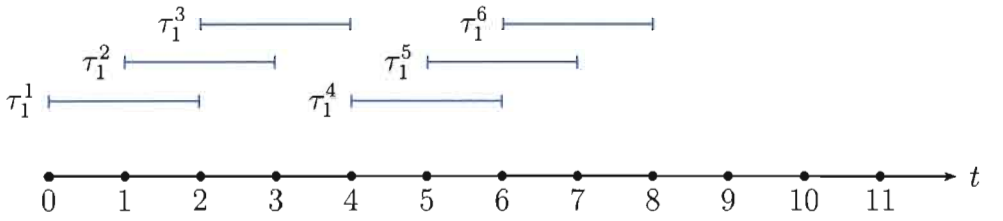

FIG. 2.3 – Fenêtre d'exécution de la tâche lourde $\tau_1 = (0, 6, 8, 8)$

Les sous-tâches $\tau_1^1\tau_1^2\tau_1^3$ forment un groupe(voir figure 2.3). En effet l'exécution de τ_1^1 à $t = 1$ oblige les autres sous-tâches s'exécuter dans le dernier slot de leurs fenêtres respectives. $D(\tau_1^1) = D(\tau_1^2) = D(\tau_1^3) = d_1^3 = 4$

Nous pouvons maintenant définir la relation d'ordre de priorité comme ce qui suit :
Soient deux tâches τ_i et τ_j dont les sous-tâches en cours d'exécution sont τ_i^k et τ_j^p. On dira que τ_i est plus prioritaire que τ_j c'est à dire $\tau_i^k \succ \tau_j^p$ si :

1. $d_i^k < d_j^p$
2. $(d_i^k = d_j^p) \wedge (b_i^k > b_j^p)$
3. $(d_i^k = d_j^p) \wedge (b_i^k = b_j^p = 1) \wedge (D_i^k > D_j^p)$

PD

Aux trois conditions utilisées par PD^2, PD ajoute deux autres conditions portant sur le facteur de charge et le bit de groupe (bit successeur de la dernière sous-tâche d'un groupe). Notons B_i^k le bit de groupe d'une sous-tâche τ_i^k.
Soient deux tâches τ_i et τ_j dont les sous-tâches en cours d'exécution sont τ_i^k et τ_j^p. On dira que τ_i est plus prioritaire que τ_j c'est à dire $\tau_i^k \succ \tau_j^p$ si :

1. $d_i^k < d_j^p$
2. $(d_i^k = d_j^p) \wedge (b_i^k > b_j^p)$
3. $(d_i^k = d_j^p) \wedge (b_i^k = b_j^p = 1) \wedge (D_i^k > D_j^p)$
4. $(d_i^k = d_j^p) \wedge (b_i^k = b_j^p = 1) \wedge (D_i^k = D_j^p) \wedge U(\tau_i) > U(\tau_j)$
5. $(d_i^k = d_j^p) \wedge (b_i^k = b_j^p = 1) \wedge (D_i^k = D_j^p) \wedge (U(\tau_i) = U(\tau_j)) \wedge (B_i^k > B_j^p)$

L'intérêt principal des ordonnancements P-équitables réside dans leur puissance d'ordonnancement. De plus, ils permettent l'utilisation d'une condition nécessaire et suffisante d'ordonnançabilité simple à évaluer. Le théorème ci-dessous illustre ces deux aspects.

1. $d_i^k < d_j^p$
2. $(d_i^k = d_j^p) \wedge (b_i^k > b_j^p)$
3. $(d_i^k = d_j^p) \wedge (b_i^k = b_j^p = 1) \wedge (\tau_i^{k+1} \succ \tau_j^{p+1})$

Exemple : Considérons deux tâches périodiques $\tau_1 = (0, 3, 5, 5)$ et $\tau_2 = (0, 2, 4, 4)$.
A $t = 0$ les sous-tâches en cours d'exécution sont τ_1^1 et τ_2^1.
$d_1^1 = \lceil \frac{1 \times 5}{3} \rceil = 2$
$d_2^1 = \lceil \frac{1 \times 4}{2} \rceil = 2$
La première condition ne permet pas de conclure, nous passons à l'étude de la deuxième condition.
$b_1^1 = \lceil \frac{1 \times 5}{3} \rceil - \lfloor \frac{1 \times 5}{3} \rfloor = 2 - 1 = 1$
$b_2^1 = \lceil \frac{1 \times 4}{2} \rceil - \lfloor \frac{1 \times 4}{2} \rfloor = 2 - 2 = 0$
La deuxième condition nous permet de dire qu'à $t = 0$ $\tau_1 \succ \tau_2$.

PD^2

Comme PF, l'agorithme PD^2 compare les sous-tâches en cours d'exécution. La différence ne se trouve qu'au niveau de la condition (3) utilisée par PF. Mais avant de présenter la définition de la relation $\succ$ par PD^2 introduisons le concept "d'échéance de groupe".

En fonction du facteur de charge d'une tâche τ_i on distingue deux catégories de tâches :
- les tâches "légères" si $0 < U(\tau_i) < \frac{1}{2}$;
- les tâches "lourdes" si $\frac{1}{2} \leq U(\tau_i) < 1$.

Lorsqu'une tâche est lourde, nous avons les propriétés suivantes :
- la longueur de la fenêtre de la première sous-tâche de chaque travail de τ_i est deux ;
- toutes les fenêtres des sous-tâches ont une longueur de deux ou trois slots

Pour les tâches lourdes, on peut définir un groupe comme étant une séquence de sous-tâches, telle que si l'ordonnancement d'une des sous-tâches se fait au dernier slot de la fenêtre, cela contraint les sous-tâches qui suivent à s'ordonnancer aux derniers slots de leurs fenêtres respectives. Pour un groupe $\tau_i^j \ldots \tau_i^k$ nous définissons D_i^k l'echéance de groupe de la sous-tâche τ_i^p avec $j \leq p \leq k$ comme ce qui suit :

$$D_i^p = \begin{cases} d_i^k - 1 & \text{si } |I_i^{k+1}| = 3 \\ d_i^k & \text{si } b_i^k = 0 \end{cases}$$

Théorème 1 ([3]) *Soit τ un système périodique à échéances sur requête et à départs simultanés, il existe un ordonnancement P-équitable sur m processeurs identiques de capacité 1 si et seulement si*

$$U(\tau) \leq m$$

Deuxième partie

Résultats du stage

Chapitre 3

Extension du contexte d'application

L'ordonnancement des systèmes de tâches périodiques à départs différés ou à échéances contraintes par les algorithmes P-équitables nécessite au préalable de reformuler les notions de base. Avant de présenter ces reformulations, nous analyserons d'abord les formules utilisées pour l'ordonnancement des tâches à départs simultanés et à échéances sur requête.

3.1 Systèmes de tâches à départs simultanés et à échéances sur requête

Pour toute tâche $\tau_i = (r_i, C_i, D_i, T_i)$, avec $r_i = 0$ et $D_i = T_i$, nous définissons :
- $f(t) = U(\tau_i) \times t$, l'ordonnancement idéal ;
- $f_{min}(t) = \lfloor U(\tau_i).t \rfloor = \lfloor f(t) \rfloor$, le minimun de temps d'exécution d'une tâche sur un processeur à un instant t ;
- $f_{max}(t) = \lceil U(\tau_i).t \rceil = \lceil f(t) \rceil$, le maximun de temps d'exécution d'une tâche sur un processeur à un instant t
- $f_{reel}(t) = \sum_{k=0}^{t-1} S(\tau_i, k)$.

Les propriétés des parties entières permettent d'établir :

$$f(t) - 1 < f_{min}(t) \leq f(t) \leq f_{max}(t) < f(t) + 1$$

La contrainte d'ordonnancement P-équitable peut être reformuler comme suit :

$$\forall t, f_{reel}(t) = f_{max}(t) \text{ ou } f_{reel}(t) = f_{max}(t)$$

Une tâche est :
- en retard si $f_{reel}(t) < f(t)$;
- en avance si $f_{reel}(t) > f(t)$;
- ponctuel si $f_{reel}(t) = f(t)$.

Nous illustrons tout cela sur la figure 3.1. Les droites "ideal", "Min" et "Max" représentent respectivement $f(t)$, $f(t)-1$ et $f(t)+1$. La courbe qui représentera $f_{reel}(t)$ devra osciller entre les droites "Min" et "Max" sans jamais les atteindre.

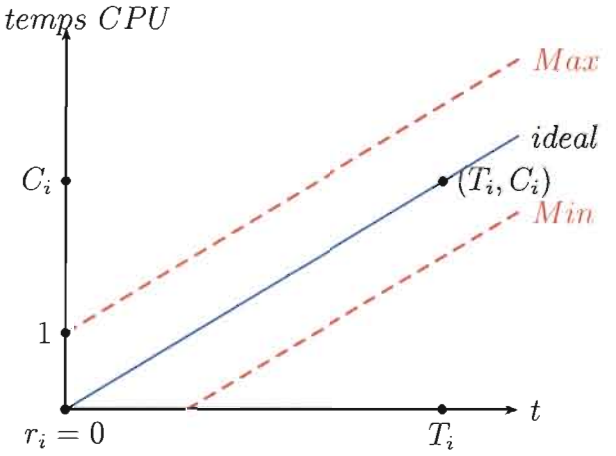

FIG. 3.1 – Ordonnancement P-équitable pour tâches périodiques synchrones à échéances sur requête

Exemple : Soit la tâche $\tau_i = (0, 3, 5, 5)$.

$$U(\tau_i) = \frac{3}{5}\,;\; f(t) = \frac{3}{5}t\,;\; f_{min}(t) = \lfloor \frac{3}{5}t \rfloor\,;\; f_{max}(t) = \lceil \frac{3}{5}t \rceil$$

t	0	1	2	3	4	5	6
$f(t)$	0.0	0.6	1.2	1.8	2.4	3.0	3.6
$f_{min}(t)$	0	0	1	1	2	3	3
$f_{max}(t)$	0	1	2	2	3	3	4
$S(\tau_i, t)$	1	0	1	1	0	1	0
$f_{reel}(t)$	0	1	1	2	3	3	4
Etat	ponctuel	avance	retard	avance	avance	ponctuel	avance

TAB. 3.1 – Exemple d'ordonnancement équitable

L'ordonnancement présenté dans le tableau 3.1 (voir figure 3.2(a)) est P-équitable alors que celui présenté dans le tableau 3.2 (voir figure 3.2(b)) ne l'est pas. En effet dans le deuxième cas, nous avons $f_{reel}(3) > f_{max}(3)$. Dans la figure 3.2 la courbe "Reel" représente $f_{reel}(t)$.

Les chaînes caractéristiques sont utilisées dans le but de faire de la prévision. En effet on remarquera que :

$$\alpha_t(\tau_i) = sign(f(t+1) - (f_{min}(t)+1))$$

t	0	1	2	3	4	5	6
$f(t)$	0.0	0.6	1.2	1.8	2.4	3.0	3.6
$f_{min}(t)$	0	0	1	1	2	3	3
$f_{max}(t)$	0	1	2	2	3	3	4
$S(\tau_i, t)$	1	1	1	0	0	1	0
$f_{reel}(t)$	0	1	2	3	3	3	4
Etat	ponctuel	avance	avance	avance	avance	ponctuel	avance

TAB. 3.2 – Exemple d'ordonnancement non équitable

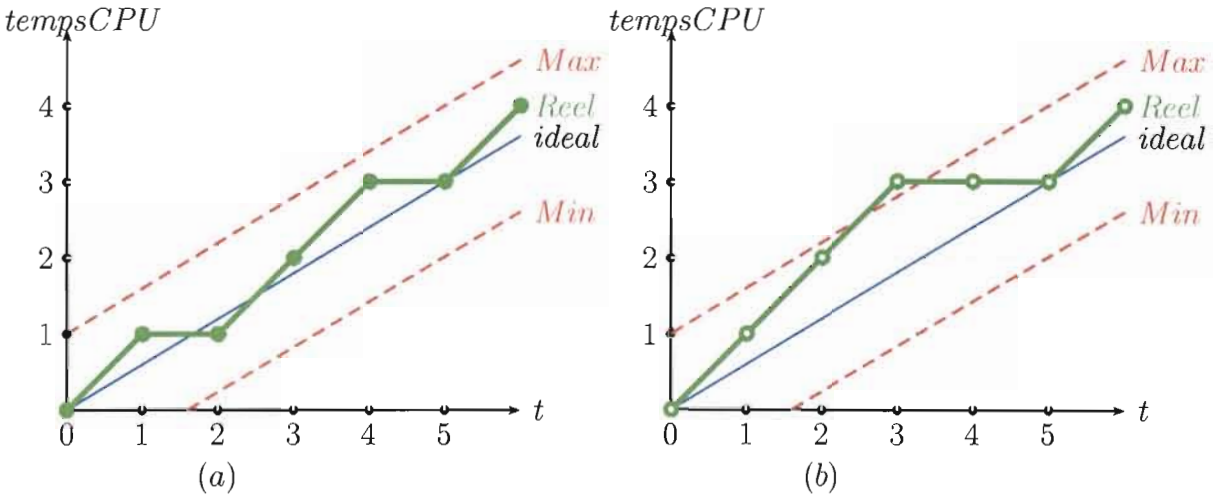

FIG. 3.2 - Exemple d'ordonnancement équitable et non équitable pour la tâche $\tau_i = (0, 3, 5, 5)$

Tout ce passait comme si on supposait qu'à une date t la tâche est en retard (c'est-à-dire qu'elle a reçu $f_{min}(t)$ de temps d'exécution sur un processeur). On cherche alors à déterminer si elle le restera à $(t + 1)$ même si elle était ordonnancée à la date t. Pour cela on détermine alors le retard à la date $(t + 1)$. Plus formellement nous avons le cheminement suivant :

- Hypothèse 1 : $f_{reel}(t) = \sum_{k=0}^{t-1} S(\tau_i, k) = \lfloor U(\tau_i).t \rfloor$
- Hypothèse 2 :$S(\tau_i, t) = 1$
- $retard_{prev}(\tau_i, t + 1) = f(t + 1) - (f_{min}(t) + 1)$

$$\alpha_t(\tau_i) = sign(retard_{prev}(\tau_i, t + 1))$$

Exemple :$\tau_i = (0, 3, 5, 5)$
Voir le tableau 3.3 et la figure 3.3.

t	0	1	2	3	4	5
$f(t+1)$	0.6	1.2	1.8	2.4	3.0	3.6
$f_{min}(t)+1$	1	1	2	2	3	4
$\alpha_t(\tau_i)$	-	+	-	+	0	-

TAB. 3.3 – Chaînes caractéristiques de $\tau_i = (0, 3, 5, 5)$

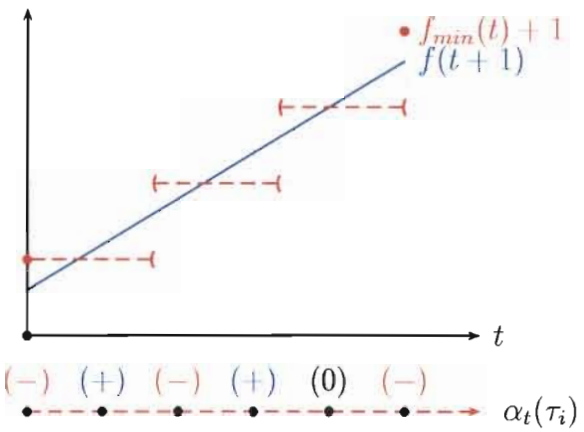

FIG. 3.3 – Chaînes caractéristiques de $\tau_i = (0, 3, 5, 5)$

De l'étude précédente, nous nous apercevons que la fonction d'ordonnancement idéalement équitable $f(t)$ pour les systèmes de tâches périodiques à départs simultanés et à échéances sur requête permet de définir le retard et les chaînes caractéristiques. Il nous suffira donc pour généraliser les concepts des algorithmes P-équitables de définir la fonction $f(t)$ pour le cas général. Nous commencerons d'abord par étudier le cas des sytèmes de tâches à départs differés et à échéances sur requête, puis celui des systèmes de tâches à départs simultanés et à échéances avant requête. Enfin nous généraliserons ces concepts pour des systèmes de tâches périodiques.

3.2 Systèmes de tâches asynchrones à échéances sur requête

3.2.1 Fonction d'ordonnancement idéalement équitable

Soit $\tau_i = (r_i, C_i, D_i, T_i)$ une tâche périodique avec $r_i > 0$ et $D_i = T_i$. Nous cherchons à déterminer l'expression de l'ordonnancement idéalement équitable.

Soit $f(t)$ cette fonction. La tâche s'active à $t = r_i$ et à la date $t = r_i + D_i$ la tâche doit avoir reçu C_i unités de temps d'exécution sur un processeur. Nous en déduisons que :

- $f(t) = 0$ si $t < r_i$;
- $f(r_i) = 0$;
- $f(r_i + T_i) = C_i$.

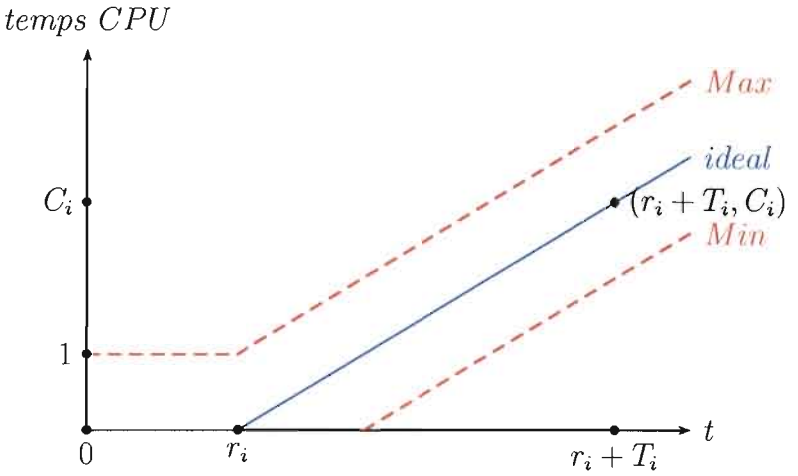

FIG. 3.4 – Ordonnancement P-équitable pour tâches périodiques asynchrones

La pente de la droite est :

$$\frac{f(r_i + T_i) - f(r_i)}{(r_i + T_i) - r_i} = \frac{C_i}{T_i}$$

Pour $t > r_i$ l'expression de $f(t)$ est : $f(t) = \frac{C_i}{T_i} \times t + const$

$$f(r_i) = 0 \Rightarrow const = -\frac{C_i.r_i}{T_i}$$

En définitive pour la tâche τ_i nous avons :

$$f(t) = \begin{cases} U(\tau_i).(t - r_i) \text{ si } t \geq r_i \\ 0 \text{ si } 0 \leq t < r_i \end{cases}$$

Nous représentons $f(t)$ par la courbe "ideal" dans la figure 3.4. Les courbes "Min" et "Max" représentent respectivement $f(t) - 1$ et $f(t) + 1$

3.2.2 Chaînes caractéristiques

L'expression de la chaîne caractéristique est :

$$\alpha_t(\tau_i) = sign(f(t+1) - (f_{min}(t) + 1)) = sign(f(t+1) - (\lfloor f(t) \rfloor + 1))$$

Pour $0 \leq t < r_i$ $\alpha_t(\tau_i) = sign(0 - (\lfloor 0 \rfloor + 1)) = -$.
Finalement :

$$\alpha_t(\tau_i) = \begin{cases} sign(U(\tau_i).(t - r_i + 1)) - (\lfloor U(\tau_i).(t - r_i) \rfloor + 1)) & \text{si } t \geq r_i \\ - & \text{si } 0 \leq t < r_i \end{cases}$$

3.2.3 Sous-tâches

La date d'activation a pour effet de translater les fenêtres de réalisation des sous-tâches de r_i. Nous aurons donc :

$$I_i^k = [r_i^k, d_i^k) \text{ avec } \begin{cases} r_i^k = r_i + \lfloor \frac{k-1}{U(\tau_i)} \rfloor \\ d_i^k = r_i + \lceil \frac{k}{U(\tau_i)} \rceil \end{cases}$$

$$b_i^k = d_i^k - r_i^{k+1} = \lceil \frac{k}{U(\tau_i)} \rceil - \lfloor \frac{k}{U(\tau_i)} \rfloor$$

Exemple : $\tau_1 = (2, 3, 5, 5)$

Voir les tableaux 3.4 et 3.5 ainsi que les figures 3.5 et 3.6.

t	0	1	2	3	4	5	6	7
$f(t+1)$	0	0	0.6	1.2	1.8	2.4	3.0	3.6
$\alpha_t(\tau_1)$	-	-	-	+	-	+	0	-

TAB. 3.4 – Chaînes caractéristiques de $\tau_1 = (2, 3, 5, 5)$

Sous-tâche	Pseudo-activation	Pseudo-échéance	Fenêtre	Bit successeur
τ_1^1	2	4	[2,4)	1
τ_1^2	3	6	[3,6)	1
τ_1^3	5	7	[5,7)	0
τ_1^4	7	9	[7,9)	1
τ_1^5	8	11	[8,11)	1
τ_1^6	10	12	[10,12)	0

TAB. 3.5 – Sous-tâches de $\tau_1 = (2, 3, 5, 5)$

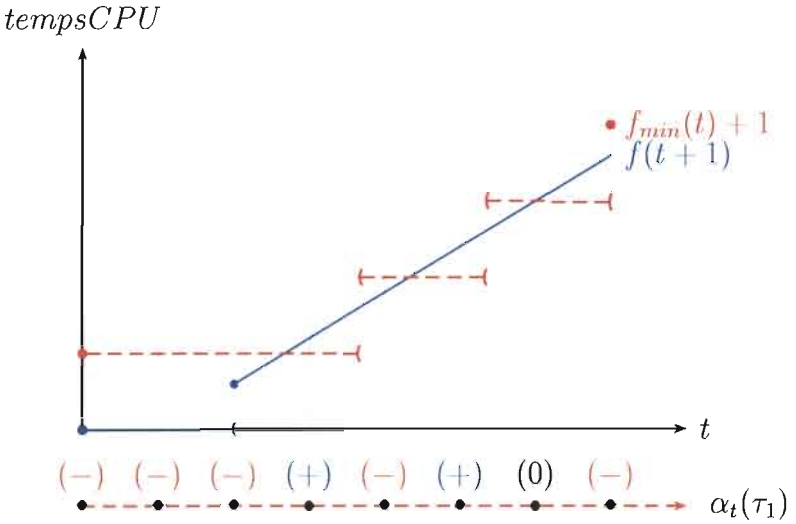

FIG. 3.5 – Chaîne caractéristique de la tâche $\tau_1 = (2, 3, 5, 5)$

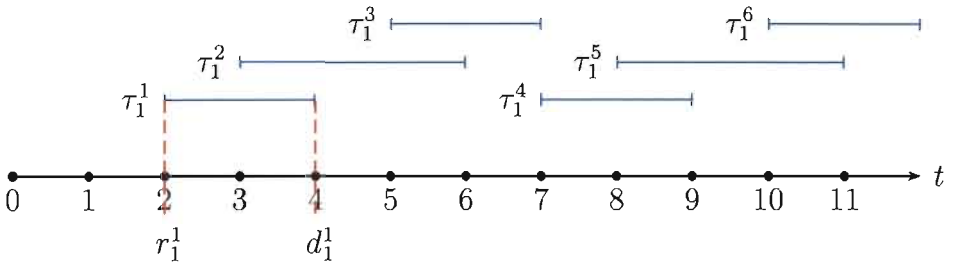

FIG. 3.6 – Fenêtres d'exécution des sous-tâches de $\tau_1 = (2, 3, 5, 5)$

3.3 Systèmes de tâches synchrones à échéances avant requête

3.3.1 Fonction d'ordonnancement idéalement équitable

Soit une tâche $\tau_i = (r_i, C_i, D_i, T_i)$ avec $r_i = 0$ et $D_i \leq T_i$. Comme dans la section précédente nous cherchons à determiner l'ordonancement idéalement équitable $f(t)$. La tâche s'active à la date $t = 0$. À $t = D_i$ et $t = T_i$ la tâche doit avoir reçu C_i unités de temps processeur. Soit :

- $f(0) = 0$
- $f(D_i) = C_i$
- $f(t) = C_i$ pour $D_i \leq t \leq T_i$

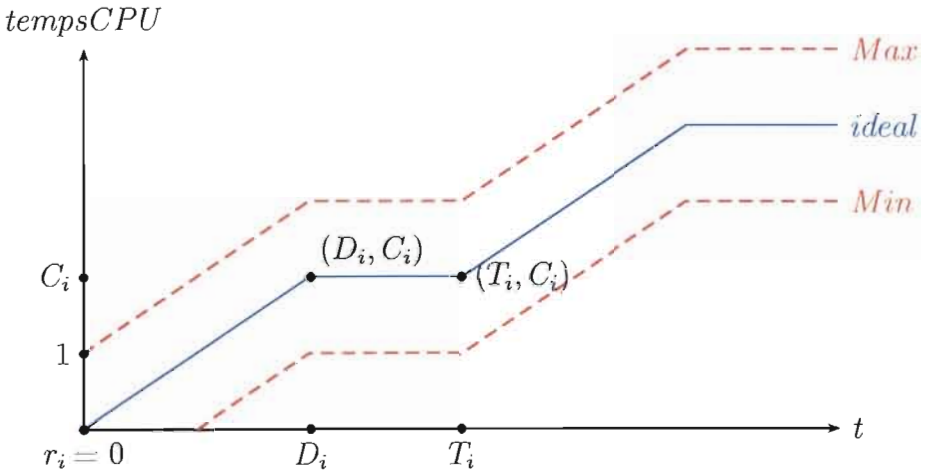

FIG. 3.7 – Ordonnancement P-équitable pour tâches périodiques synchrones à échéances avant requête

Entre $t = 0$ et $t = D_i$ nous avons un segment de droite de pente

$$\frac{f(D_i) - f(0)}{(D_i - 0)} = \frac{C_i}{D_i}$$

Pour $0 \leq t \leq D_i$ l'expression de $f(t)$ est : $f(t) = \frac{C_i}{D_i} \times t$
A chaque réactivation la tâche a accompli la somme des charges des instances précédentes.
$f(k.T_i) = k.C_i$
$\Rightarrow \frac{C_i}{D_i} \times (k.T_i) + const = k.C_i \Rightarrow const = -\frac{C_i}{D_i} \times (k.T_i) + k.C_i$
En définitive pour la tâche τ_i nous avons :

$$f(t) = \begin{cases} \frac{C_i}{D_i}.(t - k.T_i) + k.C_i & \text{si } k.T_i \leq t < k.T_i + D_i \\ (k+1).C_i & \text{si } k.T_i + D_i \leq t < (k+1).T_i \end{cases}$$

k représente l'instance de la tâche et se calcule comme suit :

$$k = \lfloor \frac{t}{T_i} \rfloor$$

Nous représentons $f(t)$ par la courbe "ideal" dans la figure 3.7. Les courbes "Min" et "Max" représentent respectivement $f(t) - 1$ et $f(t) + 1$.

3.3.2 Chaînes caractéristiques

L'expression de la chaîne caractéristique est :

$$\alpha_t(\tau_i) = sign(f(t+1) - (f_{min}(t) + 1)) = sign(f(t+1) - (\lfloor f(t) \rfloor + 1))$$

Pour $k.T_i + D_i \leq t < (k+1).T_i$
$\alpha_t(\tau_i) = sign((k+1).C_i - (\lfloor (k+1).C_i \rfloor + 1))$
$= sign((k+1).C_i - ((k+1).C_i + 1))$
$= sign(-1) = -.$

Pour $k.T_i \leq t < k.T_i + D_i$
$\alpha_t(\tau_i) = sign(\frac{C_i}{D_i}.(t - k.T_i + 1) + k.C_i) - (\lfloor \frac{C_i}{D_i}.(t - k.T_i) + k.C_i \rfloor + 1))$
$= sign(\frac{C_i}{D_i}.(t - k.T_i + 1) + k.C_i) - (\lfloor \frac{C_i}{D_i}.(t - k.T_i) \rfloor + k.C_i + 1))$ (car $k.C_i \in N$
$= sign(\frac{C_i}{D_i}.(t - k.T_i + 1) - (\lfloor \frac{C_i}{D_i}.(t - k.T_i) \rfloor + 1)).$

Finalement :

$$\alpha_t(\tau_i) = \begin{cases} sign(\frac{C_i}{D_i}.(t - k.T_i + 1) - (\lfloor \frac{C_i}{D_i}.(t - k.T_i) \rfloor + 1)) & \text{si } k.T_i \leq t < k.T_i + D_i \\ - & \text{si } k.T_i + D_i \leq t < (k+1).T_i \end{cases}$$

3.3.3 Sous-tâches

Soit $\tau_i = (0, C_i, D_i, T_i)$ une tâche périodique. Cette tâche définit une infinité d'instances $\tau_{i0}, \tau_{i1}, \ldots$
La date de démarrage d'une instance τ_{ik} est notée a_{ik}. Chaque instance τ_{ik} a C_i sous-tâches dont les dates de pseudo-activation et de pseudo-échéance sont respectivement $r_{ik}^1, r_{ik}^2, \ldots, r_{ik}^{C_i}$ et $d_{ik}^1, d_{ik}^2, \ldots, d_{ik}^{C_i}$, avec :

$$r_{ik}^p = constante1 + \lfloor \frac{p-1}{\overline{u}(\tau_i)} \rfloor$$

$$d_{ik}^{p} = constante2 + \lceil \frac{p}{\overline{u}(\tau_i)} \rceil$$

$$r_{ik}^{1} = a_{ik} \Rightarrow constante1 = a_{ik}$$

$$d_{ik}^{C_i} = a_{ik} + D_i \Rightarrow constante2 = a_{ik}$$

Nous nous ramenons au cas général de la détermination de r_i^j et de d_i^j en posant $j = kC_i + p$; $a_{ik} = kT_i$; $k = \lfloor \frac{j-1}{C_i} \rfloor$:

$$I_i^j = [r_i^j, d_i^j) \text{ avec } \begin{cases} k = & \lfloor \frac{j-1}{C_i} \rfloor \\ r_i^j = & k.T_i + \lfloor \frac{j - k.C_i - 1}{\overline{u}(\tau_i)} \rfloor \\ d_i^j = & k.T_i + \lceil \frac{j - k.C_i}{\overline{u}(\tau_i)} \rceil \end{cases}$$

$$b_i^j = d_i^j - r_i^{j+1}$$

Exemple : $\tau_i = (0, 3, 4, 5)$

Voir les tableaux 3.6 et 3.7 ainsi que les figures 3.8 et 3.9.

t	0	1	2	3	4	5
$f(t+1)$	0.75	1.5	2.25	3	3	3
$\alpha_t(\tau_i)$	-	+	+	0	-	-

TAB. 3.6 – Chaînes caractéristiques de $\tau_i = (0, 3, 4, 5)$

Sous-tâche	Pseudo-activation	Pseudo-échéance	Fenètre	Bit successeur
τ_1^1	0	2	[0,2)	1
τ_1^2	1	3	[1,3)	1
τ_1^3	2	4	[2,4)	-1
τ_1^4	5	7	[5,7)	1
τ_1^5	6	8	[6,8)	1
τ_1^6	7	9	[7,9)	-1

TAB. 3.7 – Sous-tâches de $\tau_1 = (0, 3, 4, 5)$

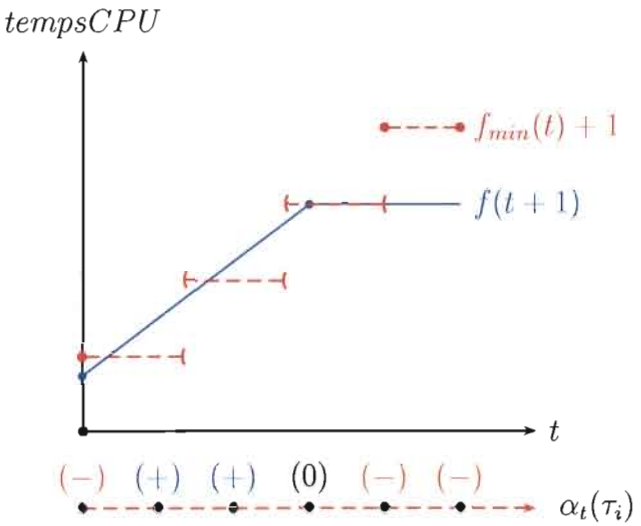

FIG. 3.8 – Chaînes caractéristiques d'une tâche à échéance avant requête : $\tau_i = (0, 3, 4, 5)$

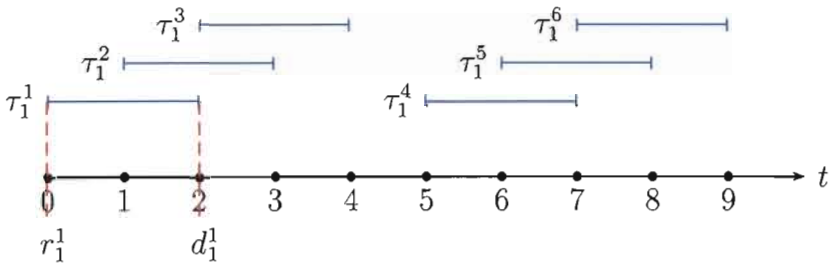

FIG. 3.9 – Sous-tâches d'une tâche à échéance avant requête : $\tau_1 = (0, 3, 4, 5)$

3.4 Généralisation

Pour généraliser tout ce qui à été présenté dans les sections précédentes nous allons diviser l'activité d'une tâche périodique $\tau_i = (r_i, C_i, D_i, T_i)$ en trois parties :

- Avant l'activation, c'est-à-dire pour $0 \leq t < r_i$.
  La fonction d'ordonnancement idéal $f(t)$ est nulle.
  La chaîne caractéristique est :
  $\alpha_t(\tau_i) = sign(f(t+1) - (\lfloor f(t) \rfloor + 1)) = sign(-1) = -$

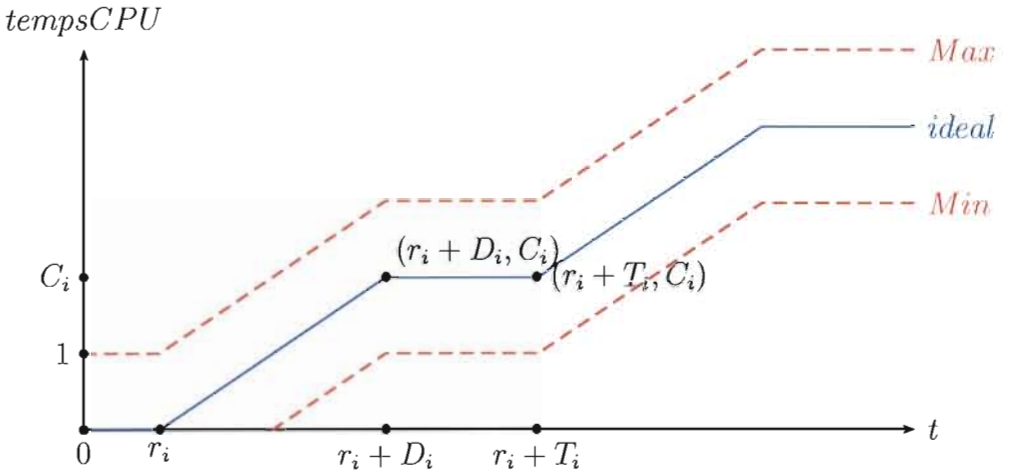

FIG. 3.10 – Ordonnancement P-équitable pour tâches périodiques

- Pendant la période d'activité, c'est-à-dire $r_i + k.T_i \leq t < r_i + k.T_i + D_i$ avec $k \in N$.
  f(t) est un segment de droite de pente C_i/D_i, avec $f(r_i + k.T_i) = k.C_i$.
  $f(t) = \frac{C_i}{D_i}t + const$
  $\frac{C_i}{D_i}(r_i + kT_i) + const = k.C_i$
  $const = -\frac{C_i}{D_i}(r_i + kT_i) + k.C_i$
  Soit que :
  $$f(t) = \frac{C_i}{D_i}(t - (r_i + kT_i)) + k.C_i$$
  La chaîne caractéristique est :
  $\alpha_t(\tau_i) = sign(f(t+1) - (\lfloor f(t) \rfloor + 1))$
  $= sign(\frac{C_i}{D_i}(t + 1 - (r_i + kT_i)) + k.C_i - (\lfloor \frac{C_i}{D_i}(t - (r_i + kT_i)) + k.C_i \rfloor + 1))$
  $= sign(\frac{C_i}{D_i}(t + 1 - (r_i + kT_i)) + k.C_i - (\lfloor \frac{C_i}{D_i}(t - (r_i + kT_i)) \rfloor + k.C_i + 1))$

$= sign(\frac{C_i}{D_i}(t+1-(r_i+kT_i)) - (\lfloor \frac{C_i}{D_i}(t-(r_i+kT_i))\rfloor + 1))$

- Pendant la période d'inacactivité, c'est-à-dire $r_i + k.T_i + D_i \leq t < r_i + (k+1).T_i$ avec $k \in N$.
  $f(t) = (k+1)C_i$ La chaîne caractéristique est :
  $\alpha_t(\tau_i) = sign(f(t+1) - (\lfloor f(t)\rfloor + 1))$
  $\alpha_t(\tau_i) = sign((k+1)C_i - (\lfloor (k+1)C_i\rfloor + 1))$
  $\alpha_t(\tau_i) = sign(-1) = -$

En résumé pour chaque tâche périodique $\tau_i = (r_i, C_i, D_i, T_i)$, à chaque instant t :

$$posons \begin{cases} k = & \lfloor \frac{t-r_i}{T_i} \rfloor \\ \overline{u}(\tau_i) = & \frac{C_i}{D_i} \\ a_{ik} = & r_i + k.T_i \\ d_{ik} = & r_i + k.T_i + D_i \end{cases}$$

Intervalle	$[0, r_i)$	$[a_{ik}, d_{ik})$	$[d_{ik}, a_{i(k+1)})$
$f(t)$	0	$\overline{u}(\tau_i)(t-a_{ik}) + k.C_i$	$(k+1)C_i$
$retard(\tau_i, t)$	0	$\overline{u}(\tau_i)(t-a_{ik}) + k.C_i - \sum_{p=0}^{t-1} S(\tau_i, p)$	$(k+1)C_i - \sum_{p=0}^{t-1} S(\tau_i, p)$
$\alpha_t(\tau_i)$	-	$sign(\overline{u}(\tau_i)(t-a_{ik}) - (\lfloor \overline{u}(\tau_i)(t-a_{ik})\rfloor + 1))$	-

TAB. 3.8 – Généralisation pour tâches périodiques

Nous représentons $f(t)$ par la courbe "ideal" dans la figure 3.10. Les courbes "Min" et "Max" représentent respectivement $f(t) - 1$ et $f(t) + 1$.

Chapitre 4

Algorithmes

Dans les chapitres précédents, nous avons parlé des algorithmes P-équitables et les extensions que nous avons apportées. Dans ce chapitre nous présenterons quelques algorithmes qui permettraient leurs implémentations. Nous avons choisi de n'implémenter que PF. Tous les exemples que nous présenterons sont obtenus en utilisant PF. Nous commencerons par donner les algorithmes utilisés pour l'ordonnancement de systèmes de tâches à départs simultanés et à échéances sur requête. Ensuite nous proposerons les algorithmes utilisés dans les contextes d'ordonnancement de tâches asynchrones et de tâches à échéances avant requête.

4.1 Systèmes de tâches à départs simultanés et à échéances sur requête

4.1.1 Description

Rappelons que pour un système de tâches périodiques à départs simultanés et à échéances sur requête, les algorithmes P-équitables (voir algorithme 4) vont :

1. déterminer les sous-ensembles de tâches urgentes, possibles et interdites ;
2. allouer les processeurs aux tâches urgentes ;
3. affecter aux processeurs libres les tâches possibles les plus prioritaires.

L'allocation des tâches aux processeurs est simulée par la fonction $Ordonnancer$ (algorithme 1). $Ordonnancer(k, \tau, t)$ alloue les k premières tâches du système τ aux processeurs libres à la date t. L'état(retard ou avance) d'une tâche ne dépendant que du signe de $retard(\tau_i, t) = U(\tau_i).t - \sum_{k=0}^{t-1} S(\tau_i, k)$, nous utilisons plutôt la formule suivante du retard :

$$retard_P(\tau_i, t) = retard(\tau_i, t) \times T_i = C_i.t - T_i \times (\sum_{k=0}^{t-1} S(\tau_i, k))$$

Si à t la tâche est exécutée sur un processeur c'est-à-dire $S(\tau_i, t) = 1$ alors :
$retard_P(\tau_i, t+1) = C_i.(t+1) - T_i \times (\sum_{k=0}^{t} S(\tau_i, k))$
$retard_P(\tau_i, t+1) = C_i.t + C_i - T_i \times (\sum_{k=0}^{t-1} S(\tau_i, k) + S(\tau_i, t))$
$retard_P(\tau_i, t+1) = retard_P(\tau_i, t) - (T_i - C_i)$
Si à t la tâche n'est pas exécutée sur un processeur c'est-à-dire $S(\tau_i, t) = 0$ alors :
$retard_P(\tau_i, t+1) = C_i.(t+1) - T_i \times (\sum_{k=0}^{t} S(\tau_i, k))$
$retard_P(\tau_i, t+1) = C_i.t + C_i - T_i \times (\sum_{k=0}^{t-1} S(\tau_i, k) + S(\tau_i, t))$
$retard_P(\tau_i, t+1) = retard_P(\tau_i, t) + (C_i)$

Algorithme 1 Ordonnancer

ENTRÉES: k ;τ ;t
SORTIES: Séquences d'ordonnancement $S(\tau_i, t)$
  $compteur \leftarrow 1$
  pour $\tau_i \in \tau$ **faire**
    si $compteur \leq (k)$ **alors**
      $S(\tau_i, t) \leftarrow 1$
      $retard(\tau_i) \leftarrow retard(\tau_i) - (T_i - C_i)$
    sinon
      $S(\tau_i, t) \leftarrow 0$
      $retard(\tau_i) \leftarrow retard(\tau_i) + C_i$
    finsi
    $compteur \leftarrow compteur + 1$
  fin pour
  Retourner $S(\tau_i, t) \forall \tau_i \in \tau$.

Une matrice de m (nombre de processeurs) colonnes et n (temps de simulation) lignes est utilisée pour enregistrer les valeurs $S(\tau_i, t)$.
Exemple : Supposons que l'ordonnancement produit la matrice

$$Sequence = \begin{pmatrix} 1 & 0 \\ 2 & 3 \\ 1 & -1 \\ 0 & 1 \end{pmatrix}$$

Le nombre de colonnes est 2, donc l'ordonnancement s'est effectué sur deux processeurs P_0 et P_1.
L'élément (1,1) de la matrice est 1, donc la tâche τ_1 s'exécute sur le processeur P_0 à la date $t = 0$.
L'élément (3,2) de la matrice est -1, donc il y a un temps creux pour le processeur P_1 à la date $t = 2$.

Algorithme 2 Compare

ENTRÉES: τ_i^k, τ_j^p
SORTIES: Sous-tache la plus prioritaire selon les conditions (1) et (2)
  Calculer d_i^k, d_j^p
  si $(d_i^k < d_j^p)$ **alors**
    Retourner $+1$
  sinon si $(d_i^k > d_j^p)$ **alors**
    Retourner -1
  finsi
  Calculer b_i^k, b_j^p
  si $(b_i^k > b_j^p)$ **alors**
    Retourner $+1$
  sinon si $(b_i^k < b_j^p)$ **alors**
    Retourner -1
  finsi
  Retourner 0.

La fonction $ComparePF$ (algorithme 3) permet de trier les tâches possibles en utilisant l'ordre de priorité défini par PF. Elle intègre $Compare$ (algorithme 2) qui compare les sous-tâches en cours d'exécution et retourne les résultats suivants :

$$Compare(\tau_i^k, \tau_j^p) = \begin{cases} +1 & \Rightarrow \tau_i^k \succ \tau_j^p \\ 0 & \Rightarrow \tau_i^k \text{ priorité égale à } \tau_j^p \\ -1 & \Rightarrow \tau_j^p \succ \tau_i^k \end{cases}$$

Lorsque deux tâches sont ex-aequo pour $ComparePF$, de façons arbitraire, nous donnons la priorité à celle qui a le plus petit numéro.

Algorithme 3 ComparePF

ENTRÉES: τ_i^k, τ_j^p
SORTIES: Sous-tache la plus prioritaire selon PF
  $c \leftarrow Compare(\tau_i^k, \tau_j^p)$
  si $c \neq 0$ **alors**
    Retourner c
  finsi
  Calculer b_i^k, b_j^p
  si $(b_i^k = 1) \wedge (b_j^p = 1)$ **alors**
    Retourner $ComparePF(\tau_i^{k+1}, \tau_j^{p+1})$
  finsi
  Retourner 0.

Algorithme 4 Algorithme P-équitable

ENTRÉES: $\tau = \{\tau_1, \ldots, \tau_n\}$;$P = \{P_1, \ldots, P_m\}$;$temp\_simulation$
SORTIES: Séquences d'ordonnancement $S(\tau_i, t)$

```
pour τ_i ∈ τ faire
  retard(τ_i) ← 0
fin pour
pour t = 0 a temp_simulation faire
  pour τ_i ∈ τ faire
    α(τ_i) ← U(τ_i).(t + 1) − ⌊U(τ_i).t⌋ − 1
    si (retard(τ_i) > 0) ∧ (α(τ_i) ≠ −) alors
      Urgent ← Urgent + τ_i
    sinon si (retard(τ_i) < 0) ∧ (α(τ_i) ≠ +) alors
      Tnegru ← Tnegru + τ_i
    sinon
      Contending ← Contending + τ_i
    finsi
  fin pour
  Trier(Contending)
  k ← nombre_tache(Urgent)
  Ordonnancer(k, Urgent, t)
  Ordonnancer(m − k, Contending, t)
  Ordonnancer(0, Tnegru, t)
fin pour
Retourner S(τ_i, t)∀τ_i ∈ τ, ∀t ∈ [0, temp_simulation).
```

4.1.2 Exemple

La figure 4.1 et le tableau 4.1 donnent un exemple de séquences d'ordonnancement sur deux processeurs produit par le système $\tau = \{\tau_0, \tau_1, \tau_2, \tau_3\}$. L'algorithme utilisé est PF, avec : $\tau_0 = (0, 2, 10, 10)$; $\tau_1 = (0, 4, 5, 5)$; $\tau_2 = (0, 1, 2, 2)$; $\tau_3 = (0, 8, 20, 20)$.
Toutes les tâches s'activent en même temps avec une fréquence d'exécution sur les processeurs qui est égale au facteur de charge $U(\tau_i)$. Par exemple la tâche τ_2 s'exécute toutes les deux unités de temps.

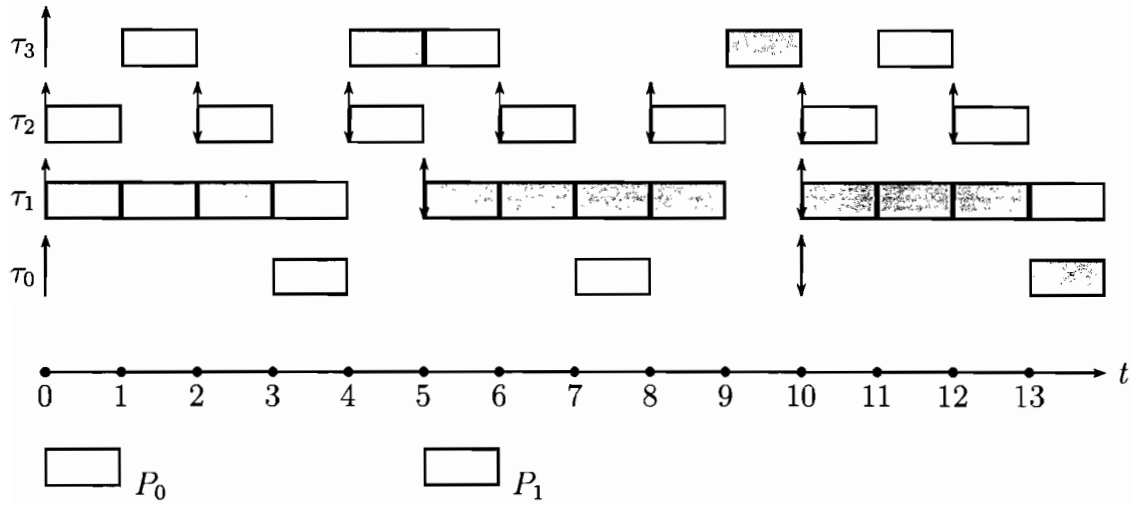

FIG. 4.1 – Séquences d'ordonnancement d'un système de tâches périodiques à départs simultanés et à échéances sur requête

	Retard				Chaine carac				Urgent	Contending	Tnegru
t	τ_0	τ_1	τ_2	τ_3	τ_0	τ_1	τ_2	τ_3			
0	0	0	0	0	-	-	-	-		$\tau_1 \succ \tau_2 \succ \tau_3 \succ \tau_0$	
1	2	-1	-1	8	-	+	0	-		$\tau_1 \succ \tau_3 \succ \tau_0$	τ_2
2	4	-2	0	-4	-	+	-	+		$\tau_1 \succ \tau_2 \succ \tau_0 \succ \tau_3$	
3	6	-3	-1	4	-	+	0	-		$\tau_0 \succ \tau_1 \succ \tau_3$	τ_2
4	-2	-4	0	12	0	0	-	0	τ_3	τ_2	τ_0, τ_1
5	0	0	-1	0	-	-	0	-		$\tau_1 \succ \tau_3 \succ \tau_0$	τ_2
6	2	-1	0	-12	-	+	-	-		$\tau_1 \succ \tau_2 \succ \tau_0$	τ_3
7	4	-2	-1	-4	-	+	0	+		$\tau_1 \succ \tau_0 \succ \tau_3$	τ_2
8	-4	-3	0	4	-	+	-	-		$\tau_1 \succ \tau_2 \succ \tau_3$	τ_0
9	-2	-4	-1	12	0	0	0	0	τ_3		τ_0, τ_1, τ_2
10	0	0	0	0	-	-	-	-		$\tau_1 \succ \tau_2 \succ \tau_3 \succ \tau_0$	
11	2	-1	-1	8	-	+	0	-		$\tau_1 \succ \tau_3 \succ \tau_0$	τ_2
12	4	-2	0	-4	-	+	-	+		$\tau_1 \succ \tau_2 \succ \tau_0 \succ \tau_3$	
13	6	-3	-1	4	-	+	0	-		$\tau_0 \succ \tau_1 \succ \tau_3$	τ_2

TAB. 4.1 – Détails de la simulation d'un système de tâches périodiques à départs simultanés et à échéances sur requête

4.2 Systèmes de tâches asynchrones à échéances sur requête

4.2.1 Description

Soit $\tau_i = (r_i, C_i, D_i, T_i)$ une tâche avec $r_i \neq 0$ et $D_i = T_i$. Pour $t \geq r_i$ une tâche est :

- **en avance** si $retard(\tau_i, t) < 0$
- **en retard** si $retard(\tau_i, t) > 0$
- **ponctuelle** si $retard(\tau_i, t) = 0$

Avant la date r_i, la tâche ne peut être exécutée sur un processeur. Pour prendre en compte cette remarque, nous dirons qu'une tâche est :

- **inactive** (noready) si $t < r_i$
- **urgente** si la tâche est en retard et que $\alpha_t(\tau_i) \neq -$;
- **interdite** (tnegru) si elle est en avance et que $\alpha_t(\tau_i) \neq +$;
- **possible** (contending) si elle n'est ni urgente ni interdite.

Les algorithmes P-équitables (voir algorithme 5) vont à chaque instant :

- répartir le système de tâches en tâches "actives" (ready) et "inactive" (noready) ;
- subdiviser les tâches actives en, tâches urgentes, interdites et possibles ;
- ordonnancer les tâches urgentes et, en fonction du nombre de processeurs libres, les tâches possibles (contending) les plus prioritaires.

4.2.2 Exemple

Dans cette section nous présentons un exemple (figure 4.2) d'exécution sur deux processeurs de l'algorithme PF pour un système de tâches périodiques à départs différés et á échéances sur requête. Le système se compose de quatres tâches à savoir $\tau_0 = (1, 2, 10, 10)$; $\tau_1 = (0, 4, 5, 5)$; $\tau_2 = (2, 1, 2, 2)$; $\tau_3 = (3, 8, 20, 20)$.
Dans le tableau 4.2 donnant les détails d'exécution, la colonne "Noready" enregistre les tâches non encore activées.

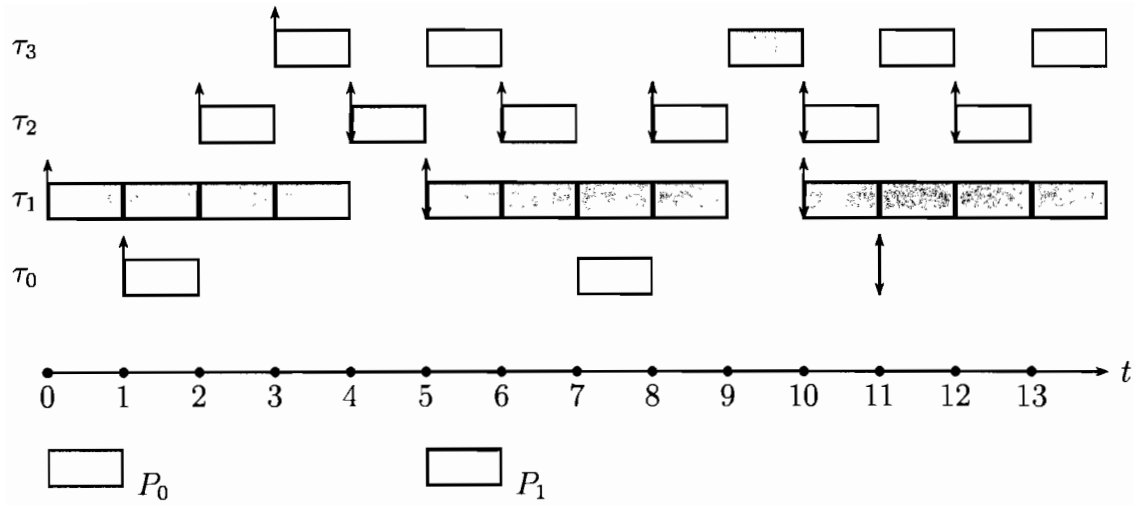

FIG. 4.2 – Séquences d'ordonnancement d'un système de tâches périodiques à départs différés et à échéances sur requête

	Retard				Chaine carac				Urgent	Contending	Tnegru	Noready
t	τ_0	τ_1	τ_2	τ_3	τ_0	τ_1	τ_2	τ_3				
0	0	0	0	0	-	-	-	-		τ_1		τ_0,τ_2,τ_3
1	0	-1	0	0	-	+	-	-		$\tau_1 \succ \tau_0$		τ_2,τ_3
2	-8	-2	0	0	-	+	-	-		$\tau_1 \succ \tau_2$	τ_0	τ_3
3	-6	-3	-1	0	-	+	0	-		$\tau_1 \succ \tau_3$	τ_0,τ_2	
4	-4	-4	0	-12	-	0	-	-		τ_2	τ_0,τ_1,τ_3	
5	-2	0	-1	-4	0	-	0	+		$\tau_1 \succ \tau_3$	τ_0,τ_2	
6	0	-1	0	-16	-	+	-	-		$\tau_1 \succ \tau_2 \succ \tau_0$	τ_3	
7	2	-2	-1	-8	-	+	0	0		$\tau_1 \succ \tau_0$	τ_2,τ_3	
8	-6	-3	0	0	-	+	-	-		$\tau_1 \succ \tau_2 \succ \tau_3$	τ_0	
9	-4	-4	-1	8	-	0	0	-		τ_3	τ_0,τ_1,τ_2	
10	-2	0	0	-4	0	-	-	+		$\tau_1 \succ \tau_2 \succ \tau_3$	τ_0	
11	0	-1	-1	4	-	+	0	-		$\tau_1 \succ \tau_3 \succ \tau_0$	τ_2	
12	2	-2	0	-8	-	+	-	0		$\tau_1 \succ \tau_2 \succ \tau_0$	τ_3	
13	4	-3	-1	0	-	+	0	-		$\tau_1 \succ \tau_3 \succ \tau_0$	τ_2	

TAB. 4.2 – Détails de la simulation d'un système de tâches périodiques à départs différés et à échéances sur requête

Algorithme 5 Algorithme P-équitable pour les tâches asynchrones

ENTRÉES: $\tau = \{\tau_1, \ldots, \tau_n\}$;$P = \{P_1, \ldots, P_m\}$;$temp\_simulation$
SORTIES: Séquences d'ordonnancement $S(\tau_i, t)$

pour $\tau_i \in \tau$ **faire**
  $retard(\tau_i) \leftarrow 0$
fin pour
pour $t = 0$ a $temp\_simulation$ **faire**
  pour $\tau_i \in \tau$ **faire**
    si $t < r_i$ **alors**
      $Noready \leftarrow Noready + \tau_i$
    sinon
      $Ready \leftarrow Ready + \tau_i$
    finsi
  fin pour
  pour $\tau_i \in Ready$ **faire**
    $\alpha(\tau_i) \leftarrow U(\tau_i).(t+1) - \lfloor U(\tau_i).t \rfloor - 1$
    si $(retard(\tau_i) > 0) \wedge (\alpha(\tau_i) \neq -$ **alors**
      $Urgent \leftarrow Urgent + \tau_i$
    sinon si $(retard(\tau_i) < 0) \wedge (\alpha(\tau_i) \neq +$ **alors**
      $Tnegru \leftarrow Tnegru + \tau_i$
    sinon
      $Contending \leftarrow Contending + \tau_i$
    finsi
  fin pour
  $Trier(Contending)$
  $k \leftarrow nombre\_tache(Urgent)$
  $Ordonnancer(k, Urgent, t)$
  $Ordonnancer(m - k, Contending, t)$
  $Ordonnancer(0, Tnegru, t)$
fin pour
Retourner $S(\tau_i, t) \forall \tau_i \in \tau, \forall t \in [0, temp\_simulation)$.

4.3 Systèmes de tâches synchrones à échéances avant requête

4.3.1 Description

Soit $\tau_i = (r_i, C_i, D_i, T_i)$ une tâche avec $r_i = 0$ et $D_i \leq T_i$.

$$posons \begin{cases} k = & \lfloor \frac{t}{T_i} \rfloor \\ \overline{u}(\tau_i) = & \frac{C_i}{D_i} \\ a_{ik} = & k.T_i \\ d_{ik} = & k.T_i + D_i \end{cases}$$

$$retard(\tau_i, t) = \begin{cases} \overline{u}(\tau_i)(t - a_{ik}) + k.C_i - \sum_{p=0}^{t-1} S(\tau_i, p) & \text{si } a_{ik} \leq t < d_{ik} \\ (k+1).C_i - \sum_{k=0}^{t-1} S(\tau_i, k) & \text{si } d_{ik} \leq t < a_{i(k+1)} \end{cases}$$

$$\alpha_t(\tau_i) = \begin{cases} sign(\overline{u}(\tau_i)(t - a_{ik}) - (\lfloor \overline{u}(\tau_i)(t - a_{ik}) \rfloor + 1)) & \text{si } a_{ik} \leq t < d_{ik} \\ - & \text{si } d_{ik} \leq t < a_{i(k+1)} \end{cases}$$

Pour $a_{ik} \leq t < d_{ik}$ une tâche est :
- **en avance** si $retard(\tau_i, t) < 0$
- **en retard** si $retard(\tau_i, t) > 0$
- **ponctuelle** si $retard(\tau_i, t) = 0$

Pour $d_{ik} \leq t < a_{i(k+1)}$, dans le cas d'un ordonnancement P-équitable la tâche ne sera pas exécutée sur un processeur. Nous dirons qu'il s'agit d'une tâche endormie.

Finalement, une tâche est :
- **endormie** (sleeping) si $d_{ik} \leq t < a_{i(k+1)}$
- **urgente** si elle est en retard et que $\alpha_t(\tau_i) \neq -$;
- **interdite** (tnegru) si elle est en avance et que $\alpha_t(\tau_i) \neq +$;
- **possible** (contending) si elle n'est ni urgente ni interdite.

Dans ce contexte un algorithme P-équitables(algorithme 6) va á chaque instant :
- répartir le système de tâches en tâches "actives"(ready) et "endormi"(sleeping) ;
- subdiviser les tâches actives en tâches urgentes, interdites et possibles ;
- ordonnancer les tâches urgentes
- ordonnancer en fonction du nombre de processeurs libres les tâches possibles(contending) les plus prioritaires.

Algorithme 6 Algorithme P-équitable pour les tâches á échéances avant requête

ENTRÉES: $\tau = \{\tau_1, \ldots, \tau_n\}$;$P = \{P_1, \ldots, P_m\}$;$temp\_simulation$
SORTIES: Séquences d'ordonnancement $S(\tau_i, t)$

pour $\tau_i \in \tau$ **faire**
  $retard(\tau_i) \leftarrow 0$
fin pour
pour $t = 0$ a $temp\_simulation$ **faire**
  pour $\tau_i \in \tau$ **faire**
    $k \leftarrow \lfloor \frac{t}{T_i} \rfloor$
    $a_{ik} \leftarrow k.T_i$
    $d_{ik} \leftarrow k.T_i + D_i$
    si $t \geq d_{ik}$ **alors**
      $Sleeping \leftarrow Sleeping + \tau_i$
    sinon
      $Ready \leftarrow Ready + \tau_i$
    finsi
  fin pour
  pour $\tau_i \in Ready$ **faire**
    $\alpha(\tau_i) \leftarrow \overline{u}(\tau_i).(t - a_{ik} + 1) - \lfloor \overline{u}(\tau_i).(t - a_{ik}) \rfloor - 1$
    si $(retard(\tau_i) > 0) \wedge (\alpha(\tau_i) \neq -$ **alors**
      $Urgent \leftarrow Urgent + \tau_i$
    sinon si $(retard(\tau_i) < 0) \wedge (\alpha(\tau_i) \neq +$ **alors**
      $Tnegru \leftarrow Tnegru + \tau_i$
    sinon
      $Contending \leftarrow Contending + \tau_i$
    finsi
  fin pour
  $Trier(Contending)$
  $k \leftarrow nombre\_tache(Urgent)$
  $Ordonnancer(k, Urgent, t)$
  $Ordonnancer(m - k, Contending, t)$
  $Ordonnancer(0, Tnegru, t)$
fin pour
Retourner $S(\tau_i, t) \forall \tau_i \in \tau, \forall t \in [0, temp\_simulation)$.

La fonction $Ordonnancer(k, \tau, t)$ alloue k tâche aux processeurs libres à la date t. La seule différence introduite par rapport à la première est le remplacement de T_i par D_i.

Algorithme 7 Ordonnancer

ENTRÉES: k ;τ ;t
SORTIES: Séquences d'ordonnancement $S(\tau_i, t)$
  $compteur \leftarrow 1$
  pour $\tau_i \in \tau$ **faire**
    si $compteur < (k)$ **alors**
      $S(\tau_i, t) \leftarrow 1$
      $retard(\tau_i) \leftarrow retard(\tau_i) - (D_i - C_i)$
    sinon
      $S(\tau_i, t) \leftarrow 0$
      $retard(\tau_i) \leftarrow retard(\tau_i) + C_i$
    finsi
    $compteur \leftarrow compteur + 1$
  fin pour
  Retourner $S(\tau_i, t) \forall \tau_i \in \tau$.

4.3.2 Exemple

La figure 4.3 et le tableau 4.4 donnent un exemple de séquence d'ordonnancement produit par un système de tâches à échéances avant requête. L'algorithme utilisé est PF. Le système de tâches est présenté dans le tableau 4.3.

Tâche	r_i	C_i	D_i	T_i	C_i/D_i	C_i/T_i
τ_0	0	3	10	12	0,3	0,25
τ_1	0	4	5	6	0,8	0,66
τ_2	0	1	2	3	0,5	0,33
τ_3	0	6	20	20	0,3	0,3
Total					1,9	1,55

TAB. 4.3 – Exemple de système de tâches à échéances avant requête

Lors de l'exécution, pour chaque instance de tâche, entre l'échéance et la réactivation, la tâche est dite en-dormie ("sleeping").

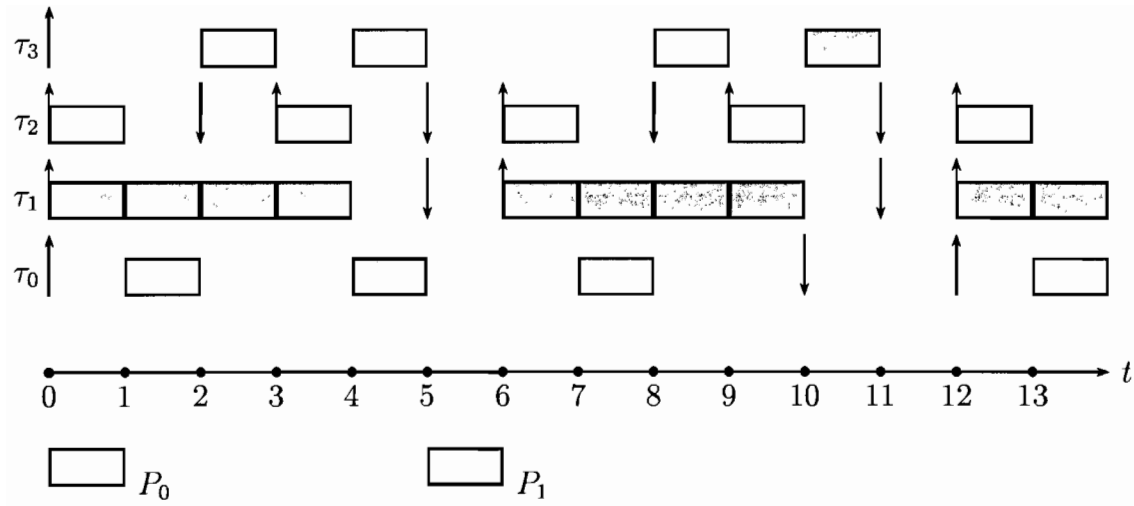

FIG. 4.3 – Séquences d'ordonnancement d'un système de tâches périodiques à départs simultanés et à échéances avant requête

	Retard				Chaine carac				Urgent	Contending	Tnegru	Sleeping
t	τ_0	τ_1	τ_2	τ_3	τ_0	τ_1	τ_2	τ_3				
0	0	0	0	0	-	-	-	-		$\tau_1 \succ \tau_2 \succ \tau_0 \succ \tau_3$		
1	3	-1	-1	6	-	+	0	-		$\tau_1 \succ \tau_0 \succ \tau_3$	τ_2	
2	-4	-2	0	12	-	+	-	-		$\tau_1 \succ \tau_3$	τ_0	τ_2
3	-1	-3	0	-2	+	+	-	+		$\tau_1 \succ \tau_2 \succ \tau_0 \succ \tau_3$		
4	2	-4	-1	4	-	0	0	-		$\tau_0 \succ \tau_3$	τ_1, τ_2	
5	-5	0	0	-10	-	-	-	-			τ_0, τ_3	τ_1, τ_2
6	-2	0	0	-4	+	-	-	+		$\tau_1 \succ \tau_2 \succ \tau_0 \succ \tau_3$		
7	1	-1	-1	2	-	+	0	-		$\tau_1 \succ \tau_0 \succ \tau_3$	τ_2	
8	-6	-2	0	8	-	+	-	-		$\tau_1 \succ \tau_3$	τ_0	τ_2
9	-3	-3	0	-6	0	+	-	0		$\tau_1 \succ \tau_2$	τ_0, τ_3	
10	0	-4	-1	0	-	0	0	-		τ_3	τ_1, τ_2	τ_0
11	0	0	0	-14	-	-	-	-			τ_3	τ_0, τ_1, τ_2
12	0	0	0	-8	-	-	-	-		$\tau_1 \succ \tau_2 \succ \tau_0$	τ_3	
13	3	-1	-1	-2	-	+	0	+		$\tau_1 \succ \tau_0 \succ \tau_3$	τ_2	

TAB. 4.4 – Détails de la simulation d'un système de tâches périodiques à départs différés et à échéances sur requête

Chapitre 5

Résultats expérimentaux

5.1 Introduction

Pour étudier les propriétés des algorithmes P-équitables, nous avons implémenté un générateur et un ordonnanceur. Le générateur génère des systèmes de tâches périodiques indépendants, puis l'ordonnanceur calcule les séquences produites par les algorithmes décrits dans le chapitre précédent. Avant de présenter les résultats obtenus nous présenterons le générateur de tâches que nous avons utilisé.

5.2 Générateur

Rappelons qu'un système τ de tâches périodiques est une collection de tâches périodiques $\tau_i = (r_i, C_i, D_i, T_i)$. L'hyper-période est le PPCM des périodes.
Exemple : $\tau = \{\tau_0, \tau_1\}$ avec $\tau_0 = (0, 2, 5, 5)$ et $\tau_1 = (2, 2, 3, 3)$.
L'hyper-période de $\tau = 15$.
Nous voulons étudier le comportement général des algorithmes P-équitables. La méthode choisie consiste à :

- générer aléatoirement un grand nombre de systèmes périodiques ;
- faire des simulations ;
- analyser les résultats statistiques obtenus.

Les simulations se font sur des durées proportionnelles aux hyper-périodes. Les hyper-périodes trop longues poseraient des problèmes de temps de calcul et de gestion de mémoires.
Pour éviter ces problèmes, notre générateur utilise la méthode présentée dans [4]. L'avantage principal de cette méthode est de limiter l'hyper-période des systèmes générés. Ainsi les séquences produites par l'ordonnanceur seront également de taille limitée.

Soit $\tau_i = (r_i, C_i, D_i, T_i)$ une tâche périodique, et $T_i = q_1^{e_1} \times q_2^{e_2} \times \ldots q_r^{e_r}$ la décomposition en produit de facteurs premiers de la période. La méthode de limitation consiste à choisir les $q_i^{e_i}$ d'une liste prédéfinie. De façon pratique ce choix se fait en utilisant les éléments d'une matrice M :

$$M = \begin{pmatrix} q_1^0 & q_1^1 & q_1^1 & q_1^2 & q_1^2 & q_1^3 & q_1^3 & q_1^4 & q_1^4 \\ q_2^0 & q_2^1 & q_2^2 & q_2^2 & q_2^3 & & & & \\ \vdots & \vdots & \vdots & \vdots & \vdots & & & & \\ q_r^0 & q_r^0 & q_r^0 & q_r^1 & q_r^1 & & & & \end{pmatrix}$$

La détermination de la période se fait en choisissant aléatoirement un indice de colonne l pour chaque ligne k de la matrice M. La période est le produit des éléments m_{kl}. Pour tout système de tâches généré, nous avons

$$\textit{hyper-periode} \leq (\prod_{i \in [1,r]} max(q_i^{e_i})).$$

L'algorithme de détermination de la période utilise les fonctions *Rand* et *Round*. La fonction $Rand(a, b)$ renvoie comme résultat un nombre réel choisi aléatoirement entre a et b et $Round(c)$ détermine l'arrondi du réel c.

Algorithme 8 CalculPeriode

ENTRÉES: matrice M
SORTIES: période T_i
  $periode \leftarrow 1$
  pour chaque ligne i de M **faire**
    $p \leftarrow Round(Rand(1, |M_i|))$
    $periode \leftarrow periode \times M_{i,p}$
  fin pour
  Retourner *periode*.

Après la détermination de la période, les autres paramètres de la tâche sont générés aléatoirement selon des lois uniformes tout en respectant les conditions

$$\begin{cases} 0 \leq C_i \leq D_i \\ 0 \leq r_i \end{cases}$$

La fonction de génération de systèmes périodiques proposé dans [4] ne permet que de générer des systèmes pour des plate-formes monoprocesseurs. Nous l'avons juste étendue en ajoutant en entrée le nombre de processeurs pour lequel le système doit être généré. En plus de la matrice M et du nombre de processeurs p, cette fonction utilise en entrée :

- les réels u_1 et u_2 avec $0 \leq u_1 \leq u_2 \leq 1$:ils permettent d'obtenir des temps d'exécutions inférieurs à T_i
- les réels o_1 et o_2 avec $0 \leq o_1 \leq o_2$:ils permettent de réguler les aspects synchrones. Si $o_1 = o_2 = 0$, les systèmes générés sont à démarrages simultanés.
- les réels d_1 et d_2 avec $0 \leq d_1 \leq d_2$:ils permettent de réguler la laxité. Si $d_1 = d_2 = 1$, les systèmes générés sont à échéances sur requête.
- le réel U : il représente la facteur de charge approximatif des systèmes
- l'entier n : il représente le nombre maximal de tâches à générer

Algorithme 9 Generateur de systèmes périodiques

ENTRÉES: matrice M, p, u_1, u_2, o_1, o_2,d_1, d_2, U, n
SORTIES: τ

$current\_load \leftarrow 0$
$\tau \leftarrow \phi)$
$i \leftarrow 0$
tantque $current\_load < U$ and $i < n$ **faire**
  $i \leftarrow i + 1$
  $T_i \leftarrow CalculPeriode(M)$
  $C_i \leftarrow max(1, Round(Rand(u_1, u_2) \times T_i))$
  $r_i \leftarrow Round(Rand(o_1, o_2) \times T_i)$
  $D_i \leftarrow Round((T_i - C_i) \times Rand(d_1, d_2)) + C_i$
  si $current\_load + \frac{C_i}{T_i} \leq p$ **alors**
    $current\_load \leftarrow current\_load + \frac{C_i}{T_i}$
    $\tau \leftarrow \tau + (r_i, C_i, D_i, T_i)$
  finsi
fin tantque
Retourner τ.

Dans notre générateur nous avons utilisé la matrice M_G définie par :

$$M_G = \begin{pmatrix} 1 & 1 & 2 & 2 \\ 1 & 1 & 1 & 3 \\ 1 & 1 & 5 & 5 \\ 1 & 1 & 7 & 7 \end{pmatrix}$$

Tous les systèmes de tâches générés auront une hyper-période limitée à 210. En effet :
$hyper - periode \leq 2 \times 3 \times 5 \times 7$
$hyper - periode \leq 210$

5.3 Systèmes de tâches à départs simultanés et à échéances sur requête

Description du test : Nous générons des systèmes de tâches périodiques à départs simultanés et à échéances sur requête. Pour différentes plate-formes de m processeurs la génération se fait en utilisant la condition $\sum \frac{C_i}{T_i} \leq m$. La simulation se fait pour $t \in [0, \textit{hyper-periode}(\tau))$

But : L'optimalité des algorithmes P-équitables ayant déjà été prouvée, le test n'est effectué que dans un but de confirmation.

Résultats

Plate-forme	Condition	Nombre de systèmes	Séquences invalides	Séquences valides
2 processeurs	$\sum \frac{C_i}{T_i} \leq 2$	5000	0	5000
3 processeurs	$\sum \frac{C_i}{T_i} \leq 3$	5000	0	5000
4 processeurs	$\sum \frac{C_i}{T_i} \leq 4$	5000	0	5000
5 processeurs	$\sum \frac{C_i}{T_i} \leq 5$	5000	0	5000
6 processeurs	$\sum \frac{C_i}{T_i} \leq 6$	5000	0	5000
TOTAL		25000	0	25000

TAB. 5.1 – Confirmation du résultat d'optimalité

5.4 Systèmes de tâches asynchrones à échéances sur requête

5.4.1 Début du cycle

Description : Nous générons des systèmes de tâches asynchrones pour des plate-formes de m processeurs respectant les conditions $\sum \frac{C_i}{T_i} \leq m$ et $\forall \tau_i, 0 < U(\tau_i) < 1$. Pour déterminer le début du cycle[1], nous utilisons les remarques suivantes :

[1]Date à partir de laquelle les séquences d'ordonnancement deviennent périodiques

- l'ordonnanceur à chaque instant t prend ses décisions en utilisant les paramètres dynamiques des tâches (le retard, la chaîne caractéristique)
- si à une date t_1 et une date t_2, tous les tâches ont exactement les mêmes paramètres dynamiques, alors les tâches qui ont été choisies à t_1 le seront à t_2

t_0 est la date d'entrée dans le cycle si et seulement si :

$$\forall t \geq t_0 \ \forall \tau_i \in \tau \begin{cases} retard(\tau_i, t) = retard(\tau_i, t+P) \\ \alpha_t(\tau_i) = \alpha_{t+P}(\tau_i) \end{cases}$$

Avec $P = hyper\text{-}periode(T_i)$.
La simulation se fait pour $t \in [0, (\max(r_i) + 3 \times P))$. Nous v érifions que la propriété citée plus haut est vérifiée pour $t \in [t_0, t_0 + P)$.

But : Le but est de pouvoir limiter la taille des séquences à produire par l'ordonnanceur.

Résultats
Dans le tableau, t_0 représente la date d'entrée dans le cycle.

Plate-forme	2	3	4	5	6	%
Nombre de systèmes	1000	1000	1000	1000	1000	
$0 \leq t_0 < \max(r_i)$	0	0	0	0	0	0%
$t_0 = \max(r_i)$	852	741	721	689	705	74,16%
$\max(r_i) < t_0 \leq \max(r_i) + P$	148	259	279	311	295	25,84%
$t_0 > \max(r_i) + P$	0	0	0	0	0	0%

TAB. 5.2 – Bornage du début du cycle pour les systèmes de tâches asynchrones à échéances sur requête

Commentaire : Nous obtenons que dans 74,16% des cas, l'entrée en cycle se fait à $t_0 = \max(r_i)$ et que dans les 25,84% des cas restants le cycle commence à $t_0 \in [\max(r_i)+1, \max(r_i)+P]$. Cela nous permet d'énoncer la conjecture suivante :

Conjecture 1 *Soit $\tau = \{\tau_1, \tau_2, \ldots\}$ un système de tâches périodiques à échéances sur requêtes et à départs différés tel que $\forall \tau_i$, $0 < U(\tau_i) < 1$. Dans un ordonnancement P-équitable, le cycle commence à une date t_0 tel que :*

$$\max(r_i) \leq t_0 \leq \max(r_i) + P$$

Avec $\tau_i = (r_i, C_i, D_i, T_i)$ et $P = PPCM(T_i)$.

5.4.2 Condition nécéssaire

Description : Nous générons des systèmes de tâches asynchrones pour des plate-formes de m processeurs respectant les conditions $m < \sum \frac{C_i}{T_i} \leq m + 1$ et $\forall \tau_i$, $0 < U(\tau_i) < 1$. En utilisant le résultat de la proposition précédente, la simulation se fait pour $0 \leq t < \max(r_i) + 2 * hyper - periode$.

But : Le but est de voir si $\sum \frac{C_i}{T_i} \leq m$ reste une condition nécessaire d'ordonnancement pour les systèmes de tâches asynchrones.

Résultats

Plate-forme	Condition	Nombre de systèmes	Séquences invalides	Séquences valides
2	$2 < \sum \frac{C_i}{T_i} \leq 3$	5000	5000	0
3	$3 < \sum \frac{C_i}{T_i} \leq 4$	5000	5000	0
4	$4 < \sum \frac{C_i}{T_i} \leq 5$	5000	5000	0
5	$5 < \sum \frac{C_i}{T_i} \leq 6$	5000	5000	0
6	$6 < \sum \frac{C_i}{T_i} \leq 7$	5000	5000	0
TOTAL		25000	25000	0

TAB. 5.3 – Etude de la condition nécessaire $\sum \frac{C_i}{T_i} \leq m$ pour les systèmes de tâches asynchrones à échéances sur requête

Commentaire : Sur des plate-formes de m processeurs, nous n'avons pas trouvé de systèmes répondant aux conditions $\sum \frac{C_i}{T_i} > m$ qui produisent des séquences valides. Nous pourrions émetre la proposition que $\sum \frac{C_i}{T_i} \leq m$ est une condition nécessaire d'ordonnancement.

Proposition 4 *Soit $\tau = \{\tau_1, \tau_2, \ldots\}$ un système de tâches périodiques à échéances sur requête et à départs différés tel que $\forall \tau_i$, $0 < U(\tau_i) < 1$. Si il existe un ordonnancement P-équitable sur m processeurs identiques de capacité 1 alors nécessairement*

$$U(\tau) \leq m.$$

Preuve : Ce résultat est indépendant de la stratégie d'ordonnancement. La preuve a été faite danc [6].

5.4.3 Condition suffisante

Test 1

Description : Nous générons des tâches asynchrones pour des plate-formes de m processeurs respectant les conditions $\sum \frac{C_i}{T_i} \leq m$ et $\forall \tau_i$, $0 < U(\tau_i) < 1$. La simulation se fait pour $0 \leq t < \max(r_i) + 2 * hyper - periode$.

But : Le but est de voir si $\sum \frac{C_i}{T_i} \leq m$ reste une condition suffisante d'ordonnancement pour les systèmes de tâches asynchrones.

Résultats

Plate-forme	Condition	Nombre de systèmes	Séquences invalides	Séquences valides
2	$\sum \frac{C_i}{T_i} \leq 2$	5000	0	5000
3	$\sum \frac{C_i}{T_i} \leq 3$	5000	0	5000
4	$\sum \frac{C_i}{T_i} \leq 4$	5000	0	5000
5	$\sum \frac{C_i}{T_i} \leq 5$	5000	0	5000
6	$\sum \frac{C_i}{T_i} \leq 6$	5000	0	5000
TOTAL		25000	0	25000

TAB. 5.4 – Première étude de la condition suffisante $\sum \frac{C_i}{T_i} \leq m$ pour les systèmes de tâches asynchrones à échéances sur requête

Commentaire : Sur une plate-forme de m processeurs, nous n'avons pas trouvé de systèmes de tâches asynchrones répondant aux conditions $\sum \frac{C_i}{T_i} \leq m$ qui produisent une séquence invalide.

Test 2

Description : Nous générons des tâches asynchrones pour des plate-formes de m processeurs respectant les conditions $\sum \frac{C_i}{T_i} = m$ et $\forall \tau_i$, $0 < U(\tau_i) < 1$. La simulation se fait pour $0 \leq t < \max(r_i) + 2 * hyper - periode$.

But : Le but est de confirmer les résultats précédents et s'assurer que $\sum \frac{C_i}{T_i} \leq m$ reste une condition suffisante d'ordonnancement pour les systèmes de tâches asyn-

chrones.

Résultats

Plate-forme	Condition	Nombre de systèmes	Séquences invalides	Séquences valides
2	$\sum \frac{C_i}{T_i} = 2$	5000	0	5000
3	$\sum \frac{C_i}{T_i} = 3$	5000	0	5000
4	$\sum \frac{C_i}{T_i} = 4$	5000	0	5000
5	$\sum \frac{C_i}{T_i} = 5$	5000	0	5000
6	$\sum \frac{C_i}{T_i} = 6$	5000	0	5000
TOTAL		25000	0	25000

TAB. 5.5 – Deuxième étude de la condition suffisante $\sum \frac{C_i}{T_i} \leq m$ pour les systèmes de tâches asynchrones à échéances sur requête

Commentaire : Sur une plate-forme de m processeurs, nous n'avons pas trouvé de systèmes de tâches asynchrones répondant aux conditions $\sum \frac{C_i}{T_i} = m$ qui produisent une séquence invalide. Nous pourrions émetre la conjecture que $\sum \frac{C_i}{T_i} \leq m$ est une condition suffisante d'ordonnancement.

Conjecture 2 *Soit $\tau = \{\tau_1, \tau_2, \ldots\}$ un système de tâches périodiques à échéances sur requête et à départs différés tel que $\forall \tau_i$, $0 < U(\tau_i) < 1$. Il existe un ordonnancement P-équitable sur m processeurs identiques de capacité 1 si et seulement si*

$$U(\tau) \leq m.$$

5.5 Systèmes de tâches synchrones à échéances avant requête

5.5.1 Date d'entrée en régime permanent

Description : Nous générons des systèmes de tâches à départs simultanés et à échéances avant requête pour des plate-formes de m processeurs respectant les conditions $\sum \frac{C_i}{T_i} \leq m$ et $\forall \tau_i$, $0 < U(\tau_i) < 1$. En utilisant le principe présenté en

(5.4.1) nous déterminons les dates d'entrée en régime permanent.

But : Nous voulons vérifier que l'entrée en cycle se fait à $t = 0$ pour des systèmes de tâches à départs simultanés et à échéances avant requête. Cela nous permettra de déterminer la taille des séquences à produire.

Résultats :
Dans le tableau, t_0 represente la date d'entrée en régime permanent.

Plate-forme	2	3	4	5	6	%
Nombre de systèmes	1000	1000	1000	1000	1000	
$t_0 = 0$	1000	1000	1000	1000	1000	100%
$0 < t_0 \leq P$	0	0	0	0	0	0%

TAB. 5.6 – Bornage de la date d'entrée en régime permanent pour les systèmes de tâches synchrones à échéances avant requête

Commentaire : Le cycle commence dès l'instant $t_0 = 0$. Nous pouvons donc limiter la taille des séquences à $[0, \textit{hyper-periode})$. Cela s'explique par le fait qu'à $t = P$ toutes les tâches sont réactivées et le système est à nouveau dans le même état qu'à $t = 0$.

5.5.2 Etude de la condition nécéssaire $\sum \frac{C_i}{T_i} \leq m$

Description : Nous générons des systèmes de tâches à départs simultanés et à échéance avant requêtes pour des plate-formes de m processeurs respectant les conditions $m < \sum \frac{C_i}{T_i} \leq m + 1$ et $\forall \tau_i$, $0 < U(\tau_i) < 1$. La simulation se fait pour $0 \leq t < hyper - periode$.

But : Le but est de voir si $\sum \frac{C_i}{T_i} \leq m$ est une condition nécessaire d'ordonnancement pour les systèmes de tâches à départs simultanés et à échéances avant requête.

Résultats

Plate-forme	Condition	Nombre de systèmes	Séquences invalides	Séquences valides
2	$2 < \sum \frac{C_i}{T_i} \leq 3$	5000	5000	0
3	$3 < \sum \frac{C_i}{T_i} \leq 4$	5000	5000	0
4	$4 < \sum \frac{C_i}{T_i} \leq 5$	5000	5000	0
5	$5 < \sum \frac{C_i}{T_i} \leq 6$	5000	5000	0
6	$6 < \sum \frac{C_i}{T_i} \leq 7$	5000	5000	0
TOTAL		25000	25000	0

TAB. 5.7 – Etude de la condition nécessaire $\sum \frac{C_i}{T_i} \leq m$ pour les systèmes de tâches synchrones à échéances avant requête

Commentaire : Sur une plate-forme de m processeurs, nous n'avons pas trouvé de systèmes répondant aux conditions $\sum \frac{C_i}{T_i} > m$ qui puissent être ordonnancé. Nous pourrions émettre la proposition que $\sum \frac{C_i}{T_i} \leq m$ est une condition nécessaire d'ordonnancement.

Proposition 5 *Soit $\tau = \{\tau_1, \tau_2, \ldots\}$ un système de tâches périodiques à échéances avant requête et à départs simultanés tel que $\forall \tau_i$, $0 < U(\tau_i) < 1$. Si il existe un ordonnancement P-équitable sur m processeurs identiques de capacité 1 alors nécessairement*

$$U(\tau) \leq m.$$

Preuve : Ce résultat est indépendant de la stratégie d'ordonnancement. La preuve a été faite danc [6].

5.5.3 Etude de la condition suffisante $\sum \frac{C_i}{T_i} \leq m$

Validité de la condition

Description : Nous générons des systèmes de tâches à départs simultanés et à échéances avant requête pour des plate-formes de m processeurs respectant les conditions $\sum \frac{C_i}{T_i} \leq m$ et $\forall \tau_i$, $0 < U(\tau_i) < 1$. La simulation se fait pour $0 \leq t < hyper - periode$.

But : Le but est de voir si $\sum \frac{C_i}{T_i} \leq m$ est une condition suffisante d'ordonnancement pour les systèmes de tâches à départs simultanés et à échéances avant requête.

Résultats

Plate-forme	Condition	Nombre de systèmes	Séquences invalides	Séquences valides
2	$\sum \frac{C_i}{T_i} \leq 2$	5000	4309	691
3	$\sum \frac{C_i}{T_i} \leq 3$	5000	4607	393
4	$\sum \frac{C_i}{T_i} \leq 4$	5000	4709	291
5	$\sum \frac{C_i}{T_i} \leq 5$	5000	4687	313
6	$\sum \frac{C_i}{T_i} \leq 6$	5000	4363	637
TOTAL		25000	22675	2325
%		25000	90,7%	9,3%

TAB. 5.8 – Etude de la condition suffisante $\sum \frac{C_i}{T_i} \leq m$ pour les systèmes de tâches synchrones à échéances avant requête

Commentaire : 90,7% des systèmes générés produit des séquences invalides. La condition $\sum \frac{C_i}{T_i} \leq m$ n'est donc plus une condition suffisante d'ordonnancement pour les systèmes de tâches à départs simultanés et à échéances avant requête.

Recherche d'une borne

Description : Nous générons des systèmes de tâches à départs simultanés et à échéance avant requête pour des plate-formes de m processeurs en respectant la condition$\forall \tau_i$, $0 < U(\tau_i) < 1$. La simulation se fait pour $0 \leq t < hyper - periode$. Au cours de la génération, la valeur maximale de $\sum \frac{C_i}{T_i}$ est diminuée progressivement.

But : Le but est de determiner une borne à partir de laquelle tous les systèmes de tâches générés produisent des séquences valides.

Résultats Les résultats de la simulation sont résumés dans le tableau 5.9.

$m = 3$ processeurs			$m = 4$ processeurs		
Condition	Valides	Invalides	Condition	Valides	Invalides
$\sum \frac{C_i}{T_i} \leq 3,0$	2,78%	97,22%	$\sum \frac{C_i}{T_i} \leq 4,0$	1,6%	98,4%
$\sum \frac{C_i}{T_i} \leq 2,7$	34,3%	65,7%	$\sum \frac{C_i}{T_i} \leq 3,6$	35,78%	64,22%
$\sum \frac{C_i}{T_i} \leq 2,4$	73,26%	26,74%	$\sum \frac{C_i}{T_i} \leq 3,2$	77,76%	22,24%
$\sum \frac{C_i}{T_i} \leq 2,1$	92,48%	7,52%	$\sum \frac{C_i}{T_i} \leq 2,8$	96,04%	3,96%
$\sum \frac{C_i}{T_i} \leq 1,8$	98,5%	1,5%	$\sum \frac{C_i}{T_i} \leq 2,4$	99,58%	0,42%%
$\sum \frac{C_i}{T_i} \leq 1,5$	99,8%	0,2%	$\sum \frac{C_i}{T_i} \leq 2,0$	99,98%	0,02%
$\sum \frac{C_i}{T_i} \leq 1,2$	100%	0%	$\sum \frac{C_i}{T_i} \leq 1,6$	100%	0%
$\sum \frac{C_i}{T_i} \leq 0,9$	100%	0%	$\sum \frac{C_i}{T_i} \leq 1,2$	100%	0%
$\sum \frac{C_i}{T_i} \leq 0,6$	100%	0%	$\sum \frac{C_i}{T_i} \leq 0,8$	100%	0%
$\sum \frac{C_i}{T_i} \leq 0,3$	100%	0%	$\sum \frac{C_i}{T_i} \leq 0,4$	100%	0%

TAB. 5.9 – Recherche d'une condition suffisante

Commentaire : La borne qui semble se dégager des résultats est $(m-1)/2$.

5.5.4 Etude de la condition nécéssaire $\sum \frac{C_i}{D_i} \leq m$

Description : Nous générons des systèmes de tâches à départs simultanés et à échéances avant requête pour des plate-formes de m processeurs respectant les conditions $m < \sum \frac{C_i}{D_i} \leq m+1$ et $\forall \tau_i, C_i < D_i$. La simulation se fait pour $0 \leq t < hyper - periode$.

But : Le but est de voir si $\sum \frac{C_i}{D_i} \leq m$ est une condition nécessaire d'ordonnancement pour les systèmes de tâches à départs simultanés et à échéances avant requête.

Résultats
Les résultats sont consignés dans le tableau 5.10.

Plate-forme	Condition	Nombre de systèmes	Séquences invalides	Séquences valides
2	$2 < \sum \frac{C_i}{D_i} \leq 3$	5000	4987	13
3	$3 < \sum \frac{C_i}{D_i} \leq 4$	5000	4889	111
4	$4 < \sum \frac{C_i}{D_i} \leq 5$	5000	4593	407
5	$6 < \sum \frac{C_i}{D_i} \leq 6$	5000	4018	908
6	$6 < \sum \frac{C_i}{D_i} \leq 7$	5000	3239	1761
TOTAL		25000	21726	3274
%		25000	86,904%	13,096%

TAB. 5.10 – Etude de la condition nécessaire $\sum \frac{C_i}{D_i} \leq m$ pour les systèmes de tâches synchrones à échéances avant requête

Commentaire : Sur des plate-formes de m processeurs, en moyenne 13% des systémes répondant aux conditions $\sum \frac{C_i}{D_i} > m$ a produit des séquences valides. $\sum \frac{C_i}{D_i} \leq m$ n'est donc pas une condition nécessaire d'ordonnancement pour des sys'tèmes de tâches périodiques à départs simultanés et à échéances avant requête.

5.5.5 Etude de la condition suffisante $\sum \frac{C_i}{D_i} \leq m$

Description : Nous générons des systèmes de tâches à départs simultanés et à échéances avant requête pour des plate-formes de m processeurs respectant les conditions $\sum \frac{C_i}{D_i} \leq m$ et $\forall \tau_i$, $C_i < D_i$. La simulation se fait pour $0 \leq t < hyper - periode$.

But : Le but est de voir si $\sum \frac{C_i}{D_i} \leq m$ est une condition suffisante d'ordonnancement pour les systèmes de tâches à départs simultanés et à échéance avant requête.

Résultats

Plate-forme	Condition	Nombre de systèmes	Séquences invalides	Séquences valides
2	$\sum \frac{C_i}{D_i} \leq 2$	5000	0	5000
3	$\sum \frac{C_i}{D_i} \leq 3$	5000	0	5000
4	$\sum \frac{C_i}{D_i} \leq 4$	5000	0	5000
5	$\sum \frac{C_i}{D_i} \leq 5$	5000	0	5000
6	$\sum \frac{C_i}{D_i} \leq 6$	5000	0	5000
TOTAL		25000	0	25000
%		25000	0%	100%

TAB. 5.11 – Etude de la condition suffisante $\sum \frac{C_i}{D_i} \leq m$ pour les systèmes de tâches synchrones à échéances avant requête

Commentaire : Tous les systèmes générés produisent des séquences valides. La condition $\sum \frac{C_i}{T_i} \leq m$ peut être proposée comme étant une condition suffisante d'ordonnancement pour les systèmes de tâches à départs simultanés et à échéances avant requête.

Conjecture 3 *Soit $\tau = \{\tau_1, \tau_2, \ldots \tau_n\}$ un système de n tâches périodiques à échéance avant requête et à départs simultanés tel que $\forall \tau_i$, $0 < U(\tau_i) < 1$ et $0 < C_i < D_i$. Il existe un ordonnancement P-équitable sur m processeurs identiques de capacité 1 si*

$$\sum_{i=1}^{n} \frac{C_i}{D_i} \leq m$$

Cette condition n'est pas nécessaire.

Conclusion

Notre stage nous a permis d'étendre le contexte d'application des algorithmes P-équitables. Nous avons ainsi redéfini la notion d'ordonnancement équitable pour des tâches à démarrages simultanés et à échéance avant requête et également pour les tâches à départs différés et à échéance sur requête. En se basant sur les résultats statistiques obtenus par simulations, des conjectures ont été émises. Cependant les preuves formelles des conjectures n'ont pas été établies, et de nombreux aspects comme celui de la cyclicité n'ont pas été étudiés. Le travail effectué lors du stage peut encore être complété.

Liste des tableaux

Table des figures

Liste des algorithmes

Bibliographie

[1] J. Goossens. *Introduction à l'ordonnancement temps réel multiprocesseur.* Proceedings of the 15th international conference on real-time systems, 2007.

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

[3] J. Goossens. *Ordonnancement temps réel multiprocesseur.* Livre, chapitre 2, 2004.

[4] J. Goossens and C. Macq. *Limitation of the hyper-period in real-time periodic task set generation.* In Teknea, editors, Proceedings of the 9th international conference on real-time systems, 2001.

[5] J. Anderson, A. Block, and A. Srinivasan. *Pfair scheduling : Beyond periodic task systems.* In Proceedings of the 12th Euromicro Conference on Real-Time Systems, pages 35-43, Juin 2000.

[6] G.C. Buttazzo. *Hard Real-time computing systems.* Kluwer Academic Publishers, 1997.

[7] S. Baruah, N. Cohen, C.G. Plaxton, and D. Varel. *Proportionate progress : A notion of fairness in resource allocation.* Algorithmica, 15 :600-625, 1996.

[8] Gillies. *Panorama de l'ordonnancement temps-reel.* Proceedings of the 4th international conference on real-time systems, 1995.

[9] S. Baruah, J. Gehrke, and C.G. Plaxton. *Fast scheduling of periodic tasks on multiple resources.* In Proceedings of th 9th International Parallel Processing Symposium, pages 280-288, Avril 1995.