Partie II : Analyse d'ordonnançabilité

Pierre-Emmanuel Hladik

INSA de Toulouse / LAAS-CNRS

5ème année TRS et M2R SAID

Plan

Présentation de l'analyse d'ordonnançabilité

Comment vérifier que les contraintes temporelles sont respectées ?

Pour qui ?

- Vérification hors-ligne : avant l'exécution pour des systèmes à contraintes strictes
- Vérification en-ligne : pendant l'exécution (test d'acceptation de nouvelles tâches) pour des systèmes à contraintes souples

Quel type d'analyse ?

- Analyse d'ordonnançabilité (schedulability test)
  vérifier si un ensemble de tâches est ordonnançable par une politique d'ordonnancement P donnée (la séquence d'ordonnancement produite par P est valide).
- Analyse de faisabilité (feasibility test)
  vérifier si un ensemble de tâches est ordonnançable (il existe une séquence d'ordonnancement valide).
- L'analyse de faisabilité est plus générale que l'analyse d'ordonnançabilité

Les méthodes d'analyse existantes

- Simulation
- Model-checking
- **Méthode analytique**
- ...

Trois approches majeures pour la « vérification analytique de l'ordonnançabilité »

- Critères basés sur les **facteurs d'utilisation et de charge**
- Critères basés sur le **pire temps de réponse**
- Critères basés sur la **demande processeur**

On ne considéra dans la suite que des tâches **indépendantes**, **synchrones**, **périodiques** à **échéances contraintes** pour des ordonnancements **préemptifs**.

Plan

Rappel (rapide) de logique : condition nécessaire et suffisante

Si P et Q sont des propriétés (assertions) :

- Q est une **condition nécessaire** de P si Q est vraie lorsque P est vraie (P⇒Q)
- Q est une **condition suffisante** de P si P est vraie lorsque Q est vraie (Q⇒P)

Exercice

- Nous avons 3 fonctions réelles : f, g et h. Pour une même valeur de x, $g(x) > 0$ est une condition nécessaire (seulement) pour $f(x) > 0$, et $g(x) > 0$ est une condition nécessaire (seulement) pour $h(x) > 0$. Par conséquent, $h(x) > 0$ est une condition **???** pour $f(x) > 0$.
- Supposons que A et B sont deux propositions. Si l'on dit que A est vraie si et seulement si B est vraie, cela veut dire que A est une condition **???** de B.
- "ab est impair" est une condition **???** pour que a+b soit pair
- A est une condition suffisante de B. Si A est faux, alors B est **???**.
- A est une condition nécessaire de B. Si A est faux, alors B est **???**.

Facteur d'utilisation

Définition : Facteur d'utilisation d'une tâche

Fraction de temps processeur requis pour l'exécution de la tâche :

$$U_i = \frac{C_i}{T_i}$$

Définition : Facteur d'utilisation d'un processeur

Fraction de temps processeur requis pour l'exécution de l'ensemble des n tâches affectées à ce processeur :

$$U = \sum_{i=1}^{n} U_i = \sum_{i=1}^{n} \frac{C_i}{T_i}$$

Théorème : Condition nécessaire de faisabilité

Un ensemble de tâches $\{\tau_i | i = 1..n\}$ *est faisable seulement si :*

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1$$

Ce théorème est indépendant de la politique d'ordonnancement

Facteur de charge

Définition : Facteur de charge d'une tâche

$$CH_i = \frac{C_i}{D_i}$$

Définition : Facteur de charge d'un processeur

$$CH = \sum_{i=1}^{n} CH_i = \sum_{i=1}^{n} \frac{C_i}{D_i}$$

Plan

Analyse sur l'utilisation du processeur

- Cette analyse se base sur le facteur d'utilisation (ou de charge) pour déterminer si une configuration de tâches est ordonnançable ou non.
- tests suffisants
- complexité faible
- difficile à étendre pour prendre en considération des contraintes additionnelles

Condition suffisante sous RM

Théorème : Condition suffisante d'ordonnançabilité sous RM [Liu et Layland, 1973]

Un ensemble de n tâches à échéance sur requête est ordonnançable par RM si :

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq n(2^{1/n} - 1)$$

- Complexité calculatoire en $O(n)$
- La borne tend vers $\ln 2 \approx 69\%$ lorsque n est grand ([Lehocky *et al.*, 1989b] montre que statistiquement cette borne est de l'ordre de 88%).
- Cas particulier : si toutes les paires de périodes de l'ensemble des tâches sont en relation harmonique, alors la borne sur U devient 1.
  Par exemple : $T_1 = 4$, $T_2 = 8$ et $T_3 = 16$

Exemple

Soit la configuration suivante de tâches, est-elle ordonnançable par la politique d'ordonnancement RM ?

	τ_1	τ_2	τ_3
C_i	32	5	4
T_i	80	40	16

Exemple

Soit la configuration suivante de tâches, est-elle ordonnançable par la politique d'ordonnancement RM ?

	τ_1	τ_2	τ_3
C_i	1	2	2
T_i	4	6	8

D'autres tests existent :

- **Condition suffisante Hyperbolic Bound [Bini *et al.*, 2001]**
- [Sha et Sathaye, 1993]
- [Manabe et Aoyagi, 1990]
- [Bini et Buttazo, 2002]
- ...

Hyperbolic Bound [Bini *et al.*, 2001]

Théorème : Hyperbolic Bound [Bini *et al.*, 2001]

Un ensemble de n tâches à échéance sur requête est ordonnançable par RM si :

$$\prod_{i=1}^{n}(U_i + 1) \leq 2$$

- Complexité en $O(n)$.
- Améliore le taux d'acceptation dans un rapport de $\sqrt{2}$ pour n grand.

Exemple

Calculer la borne de Liu et Layland puis l'hyperbolic bound pour la configuration de tâche suivante.

Condition nécessaire et suffisante sous EDF

Théorème : Condition suffisante d'ordonnançabilité [Liu et Layland, 1973]

Un ensemble de n tâches à échéance sur requête est ordonnançable par EDF si :

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1$$

- Complexité calculatoire en $O(n)$
- Condition **nécessaire et suffisante** pour les tâches à échéances sur requête.

Extension aux échéances contraintes

Théorème : Condition suffisante d'ordonnançabilité pour DM [Liu et Layland, 1973]

Un ensemble de n tâches ***à échéance contrainte*** *est ordonnançable par DM si :*

$$CH = \sum_{i=1}^{n} \frac{C_i}{D_i} \leq n(2^{1/n} - 1)$$

- Complexité calculatoire en $O(n)$

Théorème : Condition suffisante d'ordonnançabilité pour EDF [Liu et Layland, 1973]

Un ensemble de n tâches ***à échéance contrainte*** *est ordonnançable par EDF si :*

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

- N'est plus une condition nécessaire
- Complexité calculatoire en $O(n)$

Prise en considération des protocoles de ressource

Principe : ajouter le temps de blocage B_i d'une tâche τ_i à sa durée d'exécution C_i .

Théorème : Condition suffisante d'ordonnançabilité pour RM [Sha *et al.*, 1990]

Un ensemble de n tâches à échéance sur requête partageant des ressources critiques est ordonnançable par RM si :

$$\sum_{i=1}^{n} \frac{C_i + B_i}{T_i} \leq n(2^{1/n} - 1)$$

Théorème : Condition suffisante d'ordonnançabilité pour EDF [Chen et Lin, 1990]

Un ensemble de n tâches à échéance sur requête partageant des ressources critiques est ordonnançable par EDF si :

$$\sum_{i=1}^{n} \frac{C_i + B_i}{T_i} \leq 1$$

- Très pessimiste car chaque tâche va subir sa pire durée de blocage en attente de la libération d'une ressource.

Amélioration (pour RM)

Théorème : Condition suffisante d'ordonnançabilité pour RM [Sha *et al.*, 1990]

Un ensemble de n tâches à échéance sur requête partageant des ressources critiques est ordonnançable par RM si :

$$\forall i, 1 \leq i \leq n, \sum_{j \in hp_i \cup \{i\}} \frac{C_j}{T_j} + \frac{B_j}{T_j} \leq i(2^{1/i} - 1)$$

Exemple

Soit la configuration suivante de tâches, étudier son ordonnançabilité par RM.

	τ_1	τ_2	τ_3
C_i	1	1	2
T_i	2	4	8
B_i	1	1	0

Amélioration (encore)

Théorème : Condition suffisante d'ordonnançabilité pour RM [Sha *et al.*, 1990]

Un ensemble de n tâches à échéance sur requête partageant des ressources critiques est ordonnançable par RM si :

$$\sum_{i=1}^{n} \frac{C_i}{T_i} + \max_{j \in \{1,\ldots,n\}} \left\{ \frac{B_j}{T_j} \right\} \leq n(2^{1/n} - 1)$$

Serveur à scrutation (avec RM)

Théorème : Condition suffisante d'ordonnançabilité pour RM

- *Un ensemble de n périodiques tâches à échéance sur requête en présence d'un serveur à scrutation est ordonnançable par RM si :*

$$\sum_{i=1}^{n} \frac{C_i}{T_i} + \frac{C_S}{T_S} \leq (n+1)(2^{1/(n+1)} - 1)$$

- *Une tâche apériodique τ_a est ordonnançable en présence d'un ordonnancement RM (en supposant que chaque tâche dispose de son propre serveur) si :*

$$T_s + \left\lceil \frac{C_a}{C_S} \right\rceil T_s \leq D_a$$

Serveur ajournable (avec RM)

Problème : la conservation de la capacité du serveur tout au long de sa période viole une des règles fondamentales de RM qui est "la tâche prête de plus haute priorité doit immédiatement s'exécuter".

Exemple

	τ_1	τ_2	τ_3	τ_4	τ_5	τ_S
C_i	2	2	2	2	2	2
T_i	7	8				6
O_i	0	0	1	7	12	0

Conséquence : dépassement d'échéance possible pour les tâches périodiques

Théorème : Condition suffisante d'ordonnançabilité pour RM

Un ensemble de n périodiques tâches à échéance sur requête en présence d'un serveur ajournable est ordonnançable par RM si :

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq n \left(\left(\frac{U_S + 2}{2U_S + 1} \right)^{1/(n+1)} - 1 \right) \text{ avec } U_S = \frac{C_S}{T_S}$$

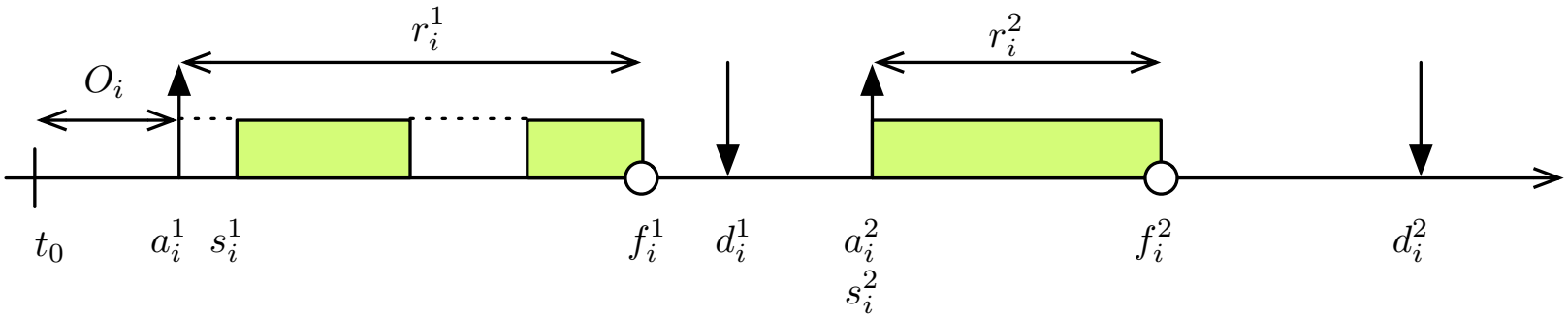
Théorème : Condition suffisante d'ordonnançabilité pour RM (2)

Un ensemble de n périodiques tâches à échéance sur requête en présence d'un serveur ajournable est ordonnançable par RM si :

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq \ln \left(\frac{U_S + 2}{2U_S + 1} \right) \text{ avec } U_S = \frac{C_S}{T_S}$$

Plan

Trajectoire et scénario d'exécution



Définition : Trajectoire du processus de tâches

La séquence des a_k^n, des c_k^n et des d_k^n définit ω une trajectoire du processus de tâches. L'ensemble des trajectoires est noté Ω.

Définition : Scénario d'exécution

Le résultat de l'ordonnancement d'un processus de tâches ω sous une certaine politique A est appelé scénario d'exécution du système.

La date $x_i^k(\omega, A)$ est la date associée à l'événement x de la k$^{\text{ième}}$ instance de la tâche τ_i pour une trajectoire ω et une politique d'ordonnancement A.

Pire temps de réponse

Définition : Pire temps de réponse (worst-case response time)

Le pire temps de réponse d'une tâche τ_i sous une politique d'ordonnancement A est le plus grand temps de réponse des instances de τ_i pour tous les scénarios d'exécution possibles sous A :

$$R_i(A) = \max_{\omega \in \Omega, k \geq 1} \{r_i^k(\omega, A)\}$$

Dans la suite, la politique d'ordonnancement ne sera pas notée.

Plan

Critère basé sur le pire temps de réponse

Principe de l'analyse par pire temps de réponse

L'analyse se fait en deux temps :

1. Calculer tous les pires temps de réponse des tâches du système.
2. Vérifier la CNS $\forall i, 1 \leq i \leq n, R_i \leq D_i$

Le calcul de R_i passe par la détermination de :

- un scénario d'exécution ω^*
- l'instance produisant le pire temps de réponse k^*

tels que :

$$R_i = r_i^{k^*}(\omega^*)$$

Parfois, il n'est pas possible d'exhiber un scénario pire cas, on construit alors un scénario $\hat{\omega}^* \notin \Omega$ qui assure :

$$R_i(\hat{\omega}^*) \geq R_i$$

La condition devient alors suffisante et on parle de solution **pessimiste**.

Instant critique

Théorème : Instant critique sous FP [Liu et Layland, 1973]

Le pire temps de réponse d'une tâche τ_i survient lorsque ***la date d'activation*** *d'une de ses instances* ***coïncide avec celle d'une instance de toutes les autres tâches plus prioritaires*** *qu'elle.*

Définition : Instant critique sous FP [Liu et Layland, 1973]

L'événement correspondant à l'activation simultanée de toutes les tâches est appelé ***instant critique****.*

Dans le cas synchrone, l'instant de démarrage est un instant critique. On a donc pour les tâches à échéances contraintes :

- ω^* qui est l'unique scénario d'exécution possible ;
- $k^* = 1$ (pour toutes les tâches) qui est la première instance de la tâche.

Calcul du pire temps de réponse sous FP

Une instance de tâche τ_i^k se termine dès que :

- elle a terminé son exécution ;
- toutes les instances des tâches plus prioritaires activées entre $[a_i^k, r_i^k)$ ont terminé leur exécution.

On a donc :

$$R_i = C_i + \sum_{j \in hp_i} n_j(R_i) C_j$$

avec hp_i l'ensemble des tâches plus prioritaires que τ_i et $n_j(t)$ le nombre de fois où la tâche τ_j est activée dans l'intervalle $[0, t)$. Pour des tâches périodiques expérimentant l'instant critique, on a :

$$n_j(t) = \left\lceil \frac{t}{T_i} \right\rceil$$

D'où :

$$R_i = C_i + \sum_{j \in hp_i} \left\lceil \frac{R_i}{T_i} \right\rceil C_j$$

Résolution de l'équation

Calcul du point fixe de la relation de récurrence :

$$R_i^{(0)} = C_i + \sum_{j \in hp_i} C_j$$

$$R_i^{(n)} = C_i + \sum_{j \in hp_i} \left\lceil \frac{R_i^{n-1}}{T_i} \right\rceil C_j$$

- Convergence assurée si : $U \leq 1$
- Complexité pseudo-polynomiale $O(n \sum_{i=1}^{n} C_i)$

Exemple

Soit la configuration suivante de tâches, calculer R_4 (avec $prio_1 > prio_2 > prio_3 > prio_4$). Confirmer le calcul par observation de la séquence d'ordonnancement.

	τ_1	τ_2	τ_3	τ_4
C_i	3	1	2	2
T_i	10	5	20	20

Prise en considération des protocoles de ressource

Ajout du temps de blocage dans l'équation :

$$R_i = C_i + B_i + \sum_{j \in hp_i} \left\lceil \frac{R_i}{T_i} \right\rceil C_j$$

Exemple

Reprendre la configuration précédente en supposant que τ_4 partage une ressource avec τ_1. La section critique a une durée de 1.

Prise en considération des serveurs apériodiques

- Au pire cas, l'interférence d'un **serveur à scrutation** est similaire à celle d'une tâche de période T_S et de durée C_S.

- L'interférence maximale d'un serveur ajournable survient pour une trajectoire où la consommation de la capacité du serveur survient à la fin d'une période et au début des suivantes, ce qui donne une interférence dans un intervalle $[0, w)$ égale à :

$$1 + \left\lceil \frac{w - C_S}{T_S} \right\rceil$$

Calcul du pire temps de réponse pour EDF

Théorème : Pire temps de réponse sous EDF

Le pire temps de réponse d'une tâche sous EDF ne survient pas nécessairement après l'instant critique.

Exemple

Soit la configuration suivante de tâches à échéance sur requête, déterminer la longueur de la période d'activité synchrone, puis R_2. Comparer.

	τ_1	τ_2
C_i	2	3
T_i	4	7

Théorème : Pire temps de réponse sous EDF [Spuri, 1996]

Le pire temps de réponse de τ_i est obtenu dans une période d'activité qui débute avec l'activation simultanée de toutes les tâches autres que τ_i

- Etude d'un ensemble de scénarios en fonction de a, retard de l'activation d'une instance de τ_i par rapport à l'instant synchrone.
- Maximum local de $R_i(a)$ pour les valeurs de a donnant lieu à un scénario tel que :
  - τ_i a son échéance qui coïncide avec celle d'une autre tâche
  - ou, toutes les tâches sont synchrones
- Algorithme pseudo-polynomial
- Calcul approché de R_i

Calcul exact dans [Leung et Merril, 1980] par simulation sur la période d'étude mais de complexité exponentielle.

Pour la prise en considération des ressources voir [Spuri, 1996].

Analyse du pire temps avec RR

Une première borne du pire temps de réponse est :

$$R_i = C_i + \sum_{j \neq i} \left\lceil \frac{R_i}{T_j} \right\rceil C_j$$

En prenant en considération les spécificités de RR, une borne moins pessimiste est donnée dans [Migge *et al.*, 2003] :

$$R_i = C_i + \min \left\{ \left\lceil \frac{C_i}{\Psi_i} \right\rceil \bar{\Psi}_i, \sum_{j \neq i} \left\lceil \frac{R_i}{T_j} \right\rceil C_j \right\}$$

avec Ψ_k la durée du quantum de temps associé à la tâche τ_k et $\bar{\Psi}_k = \sum_{j \neq k} \Psi_j$.

Une borne prenant en considération les sections critiques avec le protocole PCP est aussi proposée.

Plan

Instant oisif et période d'activité

Définition : Instant oisif (idle time)

Instant t tel qu'il n'y a plus d'instance de tâche activée avant t et non terminée à t.

Définition : Période d'activité du processeur (busy period) []

Intervalle de temps $[t_1, t_2)$ tel que :

- *t_1 et t_2 sont deux instants oisifs ;*
- *il n'y a pas d'instant oisif dans (t_1, t_2).*

Une période d'activité est un intervalle de temps $[t_1, t_2)$ pendant lequel le processeur est continuellement occupé nous avons donc :

- toutes les instances de tâche activées avant la date t_1 se sont terminées avant t_1 ;
- toutes les instances activées à ou après t_1 sont terminées à la date t_2.
- t_1 : date d'activation d'une instance
- t_2 : date de terminaison d'une instance

Instant oisif et période d'activité sous FP

Définition : Instant oisif de niveau i

Instant t tel qu'il n'y a plus d'instance de tâche de priorité supérieure ou égale à $prio_i$ activée avant t et terminée à t.

Définition : Période d'activité de niveau i (i-level busy period)

Intervalle de temps $[t_1, t_2)$ tel que :

- *t_1 et t_2 sont deux instants oisifs de niveau i ;*
- *il n'y a pas d'instant oisif de niveau i dans (t_1, t_2).*

t_2 : date de terminaison d'une instance de tâche de $prio \geq prio_i$

Exemple

Soit la configuration suivante de tâches ordonnancée selon la politique FP, représenter sur la période d'étude, la période d'activité du processeur puis ses périodes d'activité de niveau1, 2 et 3.

	τ_1	τ_2	τ_3
C_i	3	1	2
T_i	10	5	20
$prio_i$	3	2	1

Définition : Fonction de travail du processeur (processor workload)

Le travail du processeur dans l'intervalle $[0, t)$ *est la durée d'exécution des instances de tâche dont la date d'activation survient avant* t.

D'où la définition de la fonction de travail du processeur :

$$W(t) \stackrel{\text{def}}{=} \sum_{j=1}^{n} rbf(\tau_j, t) C_j \text{ avec } rbf(\tau_i, t) = \left\lceil \frac{t}{T_i} \right\rceil$$

Remarques :

- Les instances considérées ne sont pas nécessairement terminées à l'instant t.
- Si les tâches sont à départ différé, on remplace t par $t - a_i$

Définition : Fonction de travail du processeur de niveau i

Durée d'exécution cumulée des instances de tâche de priorité supérieure ou égale à $prio_i$ *dont la date d'activation survient avant* t.

$$W_i(t) = \sum_{j \in hp_i \cup \{i\}} rbf(\tau_j, t) C_j \text{ avec } rbf(\tau_i, t) = \left\lceil \frac{t}{T_i} \right\rceil$$

Définition : Fonction de demande processeur (demand bound function)

La demande processeur dans $[t_1, t_2]$ *est durée d'exécution cumulée des instances de tâche dont la date d'activation et l'échéance sont dans l'intervalle* $[t_1, t_2]$.

D'où la fonction de demande processeur :

$$dbf(t_1, t_2) = \sum_{i=1}^{n} c_i(t_1, t_2) C_i \text{ avec } c_i(t_1, t_2) = \max\left\{0, 1 + \left\lfloor \frac{t_2 - D_i}{T_i} \right\rfloor - \left\lceil \frac{t_1}{T_i} \right\rceil\right\}$$

$$dbf(0, t) = \sum_{i=1}^{n} c_i(t) C_i \text{ avec } c_i(t) = \max\left\{0, 1 + \left\lfloor \frac{t - D_i}{T_i} \right\rfloor\right\}$$

Si $D_i \leq T_i$ alors $c_i(t) = \left\lfloor \frac{t + T_i - D_i}{T_i} \right\rfloor$

Exemple

Soit la configuration suivante de tâches,

1. représenter la séquence d'ordonnancement résultant de la politique RM
2. représenter graphiquement les fonctions $W(t)$ et $dbf(t)$.
3. que représente la droite affine de pente 1 ? Quelle relation peut-elle présenter avec les autres fonctions ?

	τ_1	τ_2	τ_3
C_i	2	10	55
T_i	10	30	120
D_i	10	25	100

Plan

Principe de l'analyse par demande processeur

Test d'ordonnançabilité consistant à vérifier que toutes les requêtes devant s'exécuter dans tout intervalle de temps ne dépassent pas la capacité du processeur (c.-à-d. la longueur de l'intervalle considéré).

Demande processeur sous FP [Lehocky *et al.*, 1989b]

Théorème : Test de [Lehocky *et al.*, 1989b]

Dans un système de tâches à départ simultané, une tâche τ_i est ordonnançable si et seulement si il existe un instant $t \in [0, D_i]$ tel que $W_i(t) \leq t$.

Cela revient à rechercher la valeur minimum de la fonction $W_i(t)/t$ dans l'intervalle $[0, D_i]$

Théorème : Reformulation du test de [Lehocky *et al.*, 1989b]

Dans un système de tâches à départ simultané, une tâche τ_i est ordonnançable si et seulement si

$$\forall i, 1 \leq i \leq n, \min_{0 \leq t \leq D_i} \frac{W_i(t)}{t} \leq 1$$

Résolution du test

Les valeurs à analyser se réduit à un ensemble de points (testing set ou scheduled point test) :

$$S_i = \{kT_j | j = 1..i, k = 1..\lfloor D_i/T_j \rfloor\}$$

On cherche donc à vérifier que :

$$\forall i, 1 \leq i \leq n, \min_{t \in S_i} \frac{W_i(t)}{t} \leq 1$$

Remarquons que si une date $t \in S_i$ satisfait $W_i(t) \leq t$ alors τ_i est ordonnançable et il est inutile d'examiner d'autres points d'ordonnancement.
La complexité est pseudo-polynomiale car bornée par $\lfloor D_i/T_j \rfloor$

Exemple

soit la configuration suivante de tâches à échéance sur requête, statuer sur son ordonnançabilité par la politique RM. Mettre les calculs en relation avec le tracé de $W_i(t)$

	τ_1	τ_2	τ_3
C_i	1	2	2
T_i	4	6	8

Demande processeur pour EDF

Théorème : BHR90

Un système de tâches périodiques et à départ simultané est ordonnançable avec un facteur d'utilisation $U < 1$ si et seulement si :

$$\forall t, 0 < t < \min\{t_{lim}, H\}, dbf(0,t) \leq t$$

avec

$$t_{lim} = \frac{U}{1-U} \max_{i=1..n}\{T_i - D_i\}$$

Complexité pseudo-polynomiale $O(n \max_{i=1..n}\{T_i - D_i\})$

Bibliographie


N. C. Audsley,
Optimal Priority Assigment and Feasibility of Static Priority Tasks With Arbitrary Start Times.
Rapport interne Dept. Computer Science, University of York, YCS 164, 1991.


S. K. Baruah.
Dynamic- and Static-priority Scheduling of Recurring Real-time Tasks.
Real-Time Syst., 24(1) :93–128, 2003.


E. Bini, G. C. Butazzo et G. M. Butazzo.
A hyperbolic bound for the rate monotonic algorithm.
Proceedings of the 13th Euromicro Conference on Real-Time Systems, 59–66, 2001.


E. Bini et G. Butazzo.
The space of rate monotonic schedulability.
Proceedings of the 23rd Real-Time Systems Symposium, 2002.


G. C. Buttazzo.
Hard real-time computing systems : predictable scheduling algorithms and applications.
Kluwer Academic Publishers, 2002.


Giorgio C. Buttazzo.
Rate monotonic vs. EDF : judgment day.

Real-Time Syst., 29(1) :5–26, 2005

A. Burns, N. Hayes et M. F. Richardson
Generating feasible cyclic schedules.
Control Engineering and Practice, 3(2) :151–162, 1995.

M. Chen et K. Lin.
Dynamic priority ceilings : a concurrency protocol for real-time systems.
The Journal of Real-Time Systems, 2(4), 1990.

A. Colin, I. Puaut, C. Rochange et P. Sainrat.
Calcul de majorants de pire temps d'exécution : état de l'art.
Rapport interne IRISA, PI 1461, 2003.

M. L. Dertouzos.
Control Robotics : The Procedural Control of Physical Processes.
Proceedings of IFIP Congress, 807–813, 1974

J.-P. Elloy.
Editorial : Quelle informatique est donc nécessaire pour automatiser en temps réel.
Technique et Science Informatiques, 7(5) :395–396, 1988.

L. George.
Ordonnancement en-ligne temps réel critique dans les systèmes distribués.
Thèse de doctorat en informatique, Université de Versaille St Quentin, 1998.


L. George.
Conditions de faisabilité pour l'ordonnancement temps réel préemptif et non préemptif.
Ecole d'Eté Temps Réel, 135–150, 2005.


K. Jeffay, D. F. Stanat et C. U. Martel.
On non-preemptive scheduling of periodic and sporadic tasks.
Prooceedings of IEEE Real-Time Systems Symposium, 129–139, 1991.


J. P. Lehoczky, L. Sha et J. K. Strosnider
Enhanced aperiodic responsiveness in hard real-time environments.
Proceedings of IEEE Real-Time Systems Symposium, 1989.


J. P. Lehoczky, L. Sha L. et Y. Ding
The Rate-Montonic scheduling algorithm : exact characterization and average case behaviour.
Proceedings of the IEEE Real-Time Systems Symposium, 166–17, 1989.


J. Leung et M. Merril
A note on preemptive scheduling of periodic real-time tasks.
Information Processing Letters, 11(3), nov. 1980.


J. Leung et J. Whitehead.
On the complexity of fixed-priority scheduling of periodic, real-time tasks.
Performance Evaluation, 2 :237–250, 1982.


C. L. Liu et J. W. Layland.
Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment.
Journal of the ACM, 20(1),46–61, 1973.


C. D. Locke.
Software architecture for hard real-time applications : cyclic executives vs. fixed priority executives.
Real-Time Syst., 4(1) :37–53, 1992.


Y. Manabe et S. Aoyagi.
A feasibility decision algorithm for Rate Monotonic and Deadline Monotonic scheduling.
Real-Time Systems, 14 :171–181, 1998.


J. Migge, A. Jean-Marie et N. Navet
Timing analysis of compound scheduling policies : application to Posix1003.1b
Journal of Scheduling, 6(5) :457-482, 2003.


A. K. Mok.
Fundamental Design Problems for the Hard Real-Time Environmen.
PhD. Massachussets Institute of technology, 1983.


M.L. Dertouzos et A.K Mok.
Multiprocessor Online Scheduling of Hard-Real-Time Tasks.
IEEE Transactions on Software Engineering, 15(12) :1497–1506, 1989.


N. Navet et J. M. Migge.
Fine Tuning the Scheduling of Tasks on Posix1003.1b Compliant Systems.
Rapport interne INRIA Lorraine, RR-3730, 2003.


L. Sha, R. Rajkumar et J. P. Lehoczky.
Priority inheritance protocol : an approach to real-time synchronization.
IEEE Transactions on Computers, 39(9) :1175–1185, 1990.


L. Sha et S. S. Sathaye.
A systematic approach to designing distributed real-time systems.
IEEE Computer, 26 :68–78, 1993.


M. Spuri
Analysis of deadline scheduled real-time systems.
INRIA Research Rapport, No. 2772, 1996.


J. K. Strosnider, J. P. Lehoczky, L. Sha
The deferrable server algorithm for enhancing aperiodic responsiveness in hard real-time environments.
IEEE Transactions on Computers, 4(1), 1995.


T. S. Tia, J. Liu et M. Shankar
Algorithms and Optimality of Scheduling Soft Aperiodic Requests in Fixed-Priority Preemptive Systems.
Real-Time Systems Journal, 10(1) :23–43, 1996.


J. Xu et D. L. Parnas.
Priority Scheduling Versus Pre-Run-Time Scheduling.
Real-Time Syst., 18(1) :7–23, 2000.