Ordonnancement multiprocesseur

Joël GOOSSENS

Université Libre de Bruxelles

ETR'07 — 6 septembre 2007

Principes généraux

- Nous considérons l'ordonnancement de n tâches périodiques/sporadiques : $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$
- Sur une plate-forme parallèle composée de m processeurs identiques : $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_m$

Parallélisme limité

- L'ordonnancement sur une architecture multiprocesseur devra respecter les contraintes suivantes :
  - un processeur exécute au plus une tâche à chaque instant ;
  - une tâche s'exécute sur au plus un processeur à chaque instant.

Discipline neuve

- La théorie de l'ordonnancement pour les systèmes temps réel monoprocesseurs est un domaine de recherche bien établi à ce jour, il y a de nombreux travaux solides qui couvrent largement ce domaine de recherche.
- En revanche, l'ordonnancement pour les systèmes temps réel multiprocesseurs est une domaine de recherche relativement neuf, peu de résultats existent à ce jour.

Taxonomie des plates-formes multiprocesseurs I

- Nous pouvons distinguer au moins les types de plates-formes suivantes (du moins général au plus général)
  - Identiques. Plate-forme multiprocesseur constituée de plusieurs processeurs identiques, dans la mesure où ils sont interchangeables et ont la même puissance de calcul.
  - Uniforme. En revanche, chaque processeur d'une plate-forme uniforme est caractérisé par sa capacité de calcul, avec l'interprétation suivante : lorsqu'un travail s'exécute sur un processeur de capacité de calcul s pendant t unités de temps il réalise $s \times t$ unités de travail.
  - Spécialisée. Pour ce type de plate-forme on définit un taux d'exécution $r_{i,j}$ associé à chaque couple travail-processeur (J_i, P_j), avec l'interprétation suivante : J_i réalise $(r_{j,i} \times t)$ unités de travail lorsqu'il s'exécute sur le processeur P_i pendant t unités de temps.
- Notons que Identique $\subset$ Uniforme $\subset$ Spécialisée.

Taxonomie des plates-formes multiprocesseurs II

- Les plates-formes identiques sont donc des plates-formes homogènes, les plates-formes uniformes et indépendantes sont des plates-formes hétérogènes.
- Dans la suite, nous allons considérer le cas particulier des plates-formes identiques composées de m processeurs : $P_1, P_2, \ldots, P_m$.

Familles d'algorithmes multiprocesseurs I

- On distingue en général deux types d'approche pour l'ordonnancement multiprocesseur :
  - Partitionnement. Il s'agit de partitionner l'ensemble des n tâches en m sous-ensembles disjoints : $\tau^1, \tau^2, \ldots, \tau^m$ et d'ordonnancer ensuite l'ensemble τ^i sur le processeur $\mathcal{P}_i$ avec une stratégie d'ordonnancement locale monoprocesseur. Les tâches ne sont donc pas autorisées à migrer d'un processeur à l'autre.

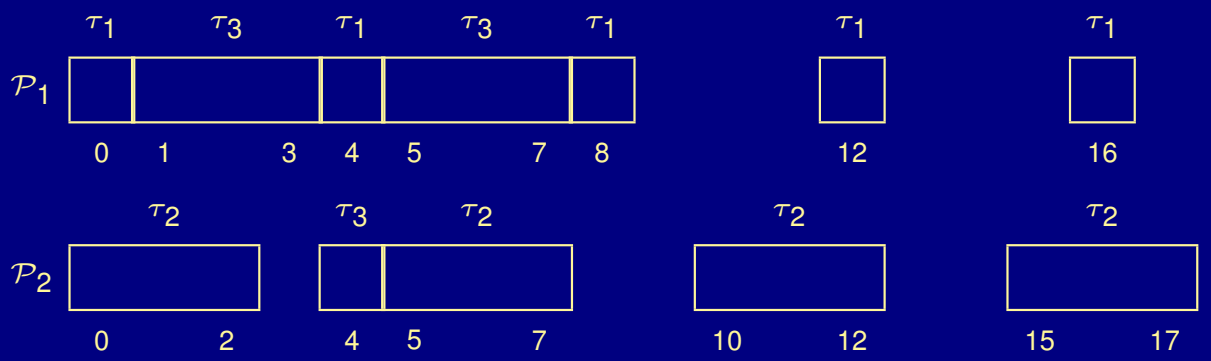
Familles d'algorithmes multiprocesseurs II

- Stratégie globale. Il s'agit d'appliquer globalement sur l'entièreté de la plate-forme multiprocesseur une stratégie d'ordonnancement et d'attribuer à chaque instant les m processeurs aux m tâches/travaux les plus prioritaires. Par exemple est utilisant RM ou EDF. Dans ce cas, outre la préemption des tâches, on autorise aussi la migration de ces dernières ; c.-à-d. qu'une tâche peut commencer son exécution sur un processeur $\mathcal{P}_j$, être préemptée et reprendre son exécution ultérieurement sur un autre processeur (disons $\mathcal{P}_i$, $i \neq j$).

Un exemple d'ordonnancement pour « global DM »

	C	T	D
τ_1	1	4	4
τ_2	3	5	5
τ_3	4	20	20

$$U(\tau) = 1.05$$
$$U_{\max}(\tau) = 0.6$$



Familles incomparables — Définition

- Ces deux familles d'algorithmes sont incomparables :
  - il existe des systèmes ordonnançables avec un partitionnement et pour lesquels les approches globales échouent et
  - il existe des systèmes ordonnançables avec une stratégie globale pour lesquels aucun partitionnement n'est ordonnançable.

Familles incomparables — Exemple

Exemple (Leung 82)

Le système τ composé des quatre tâches suivantes :

	C	T	D
τ_1	1	2	2
τ_2	2	4	4
τ_3	2	3	3
τ_4	2	6	6

$U(\tau) = 2$
$U_{\max}(\tau) = 0.666\ldots$

est ordonnançable sur deux processeurs par la partition $\tau^1 = \{\tau_1, \tau_2\}, \tau^2 = \{\tau_3, \tau_4\}$. Cependant, peu importe les priorités statiques utilisées, on n'arrive pas à ordonnancer τ sur deux processeurs.

Familles incomparables — Exemple II

À présent, considérons le système τ' composé des trois tâches suivantes :

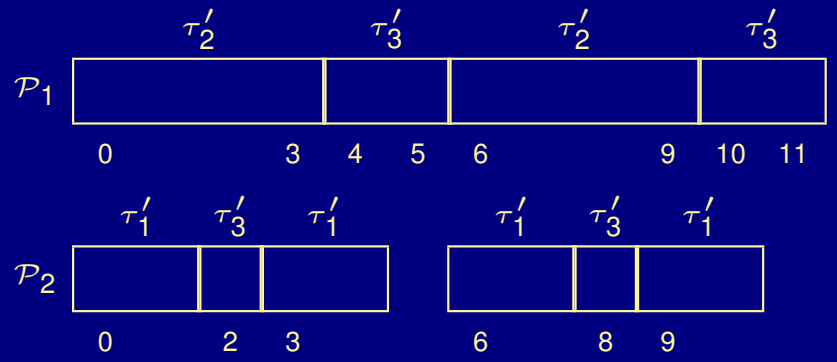
Exemple (Leung 82)

	C	T	D
τ'_1	2	3	3
τ'_2	4	6	6
τ'_3	6	12	12

$$U(\tau') = 1.833\ldots$$
$$U_{\max}(\tau') = 0.666\ldots$$

Avec les priorités $\tau'_1 > \tau'_2 > \tau'_3$ le système est ordonnançable sur deux processeurs :

Ordonnancement de τ'



Cependant, il ne peut pas être partitionné en deux groupes (ou moins), car $U(\tau'_i) + U(\tau'_j) > 1 \qquad \forall i \neq j$.

Techniques par partitionnement $\Rightarrow$ Bin Packing

- Le problème de partitionnement est en fait un problème bien connu sous le nom de Bin Packing dans lequel il faut placer dans m boîtes de tailles identiques, k objets de tailles différentes.
- Dans notre cas, les objets sont les tâches, leurs tailles sont leurs utilisations $U(\tau_i)$, la taille de chaque boîte est l'utilisation maximale que l'on peut atteindre : $\ln 2$ pour RM, 1 pour EDF.
- Ce problème est NP-Difficile, on utilise dès lors des heuristiques comme next-fit, best-fit, etc. pour approcher la solution optimale.

Rate Monotonic en partitionné I

- Pour RM partitionné, il semble naturel d'utiliser la condition $U(\tau) < n_p(2^{1/n_p} - 1)$ afin de vérifier l'ordonnançabilité des tâches assignées au processeur p (où n_p est le nombre de tâches assignées au processeur P).
- Cependant la borne n'est pas assez « serrée » pour définir une bonne stratégie multiprocesseur. On peut en effet montrer que l'on peut perdre jusqu'à 50 % des ressources.

Rate Monotonic en partitionné II

Exemple (Andersson 03)

Considérons $m + 1$ tâches avec $T_i = 1$ et $C_i = \sqrt{2} - 1 + \epsilon$ pour une plate-forme composée de m processeurs. Il y a nécessairement un processeur p qui accepte deux tâches. Sur ce processeur, l'utilisation est $2 \cdot (\sqrt{2} - 1 + \epsilon)$ qui est plus grand que $2 \cdot (\sqrt{2} - 1)$. Par conséquence, il n'y a pas de partition qui garantisse l'ordonnançabilité avec le test choisi. En considérant le cas limite $\epsilon \longrightarrow 0$ et $m \longrightarrow \infty$ nous pouvons voir que à la limite on ne peut pas garantir l'ordonnançabilité au dessus de $\sqrt{2} - 1$, i.e., approximativement 41 %.

EDF en partitionné : FFDU I

- LIU et LAYLAND ont montré que des tâches à échéance sur requête sont ordonnançables par EDF sur un processeur si et seulement si $U(\tau) \leq 1$.
- LOPEZ et al. ont montré que l'utilisation de cette borne monoprocesseur peut être utilisée pour définir une stratégie de partitionnement multiprocesseur : *First-Fit-Decreasing-Utilization* (FFDU). I.e., que dans le problème de *bin packing* on choisit à chaque étape la première boîte qui convient en considérant les tâches par valeurs décroissantes du facteur d'utilisation.
- Les mêmes chercheurs ont défini la condition d'ordonnançabilité suivante :

EDF en partitionné : FFDU II

Théorème (FFDU EDF Utilisation)

τ un ensemble de tâches périodiques avec échéance sur requête, est ordonnançable par l'une des partitions FFDU *(et* EDF *localement sur chaque processeur) si*

$$U(\tau) < (m+1)/2 \qquad \textit{et} \qquad U_{\max} \leq 1$$

Techniques globales – Optimalité impossible

Le premier résultat important, mais malheureusement négatif, est l'absence d'algorithme optimal en ligne, mais d'abord une définition.

Définition (Algorithme en ligne)

Les algorithmes d'ordonnancement en ligne prennent leurs décisions à chaque instant sur base des caractéristiques des travaux actifs arrivés jusque là, sans connaissance des travaux qui arriveront dans le futur.

Théorème (Hong 88)

Pour tout $m > 1$, aucun algorithme d'ordonnancement en ligne et optimal ne peut exister pour des systèmes avec deux ou plus d'échéances distinctes.

Anomalies d'ordonnancement – Définition

Définition (Anomalie)

Nous disons qu'un algorithme d'ordonnancement souffre d'anomalies si un changement qui est intuitivement positif dans un système ordonnançable peut le rendre non ordonnançable.

Définition (Changement intuitivement positif)

C'est un changement qui diminue le facteur d'utilisation d'une tâche comme augmenter la période, diminuer un temps d'exécution (ce qui est équivalent à augmenter la vitesse des processeurs). Cela peut être aussi de donner plus de ressources au système comme ajouter des processeurs.

Anomalies d'ordonnancement – Exemple I

- Les algorithmes d'ordonnancement multiprocesseurs sont sujet à des anomalies.
- Par exemple, pour les algorithmes à priorité fixe, il existe des systèmes qui sont ordonnançables avec une assignation de priorité mais lorsque une période est augmentée et les priorités conservées ces systèmes ne sont plus ordonnançables !

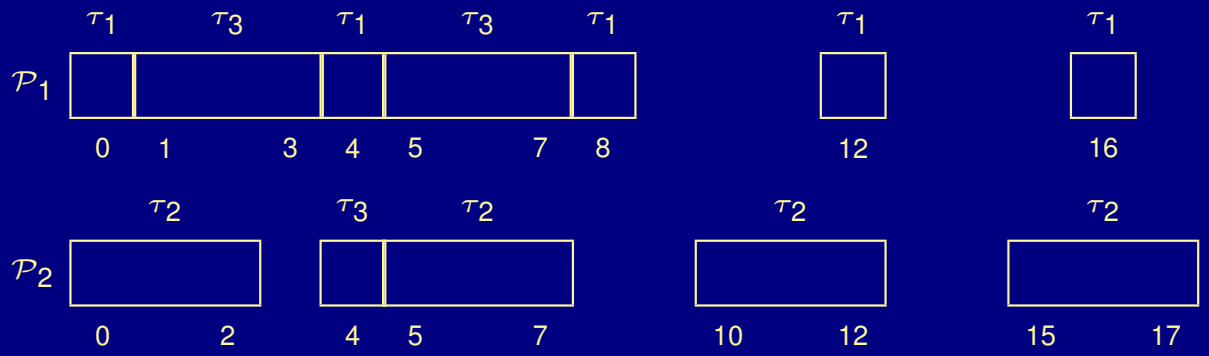
Anomalies d'ordonnancement – Exemple II

C'est le cas du système

	C	T	D
τ_1	1	4	2
τ_2	3	5	3
τ_3	7	20	8

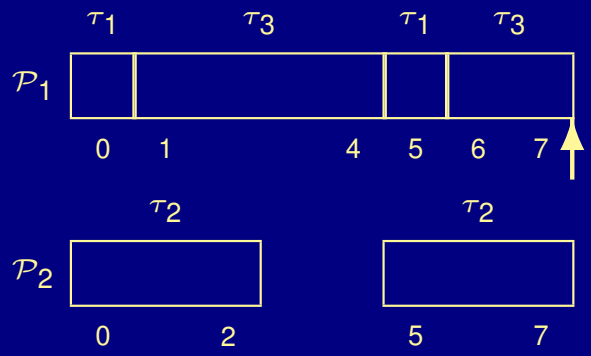
$U(\tau) = 1.2$
$U_{\max}(\tau) = 0.6$

Ce système est ordonnançable sur 2 processeurs pour le « global DM » $(\tau_1 \succ \tau_2 \succ \tau_3)$



Anomalies d'ordonnancement – Exemple III

En revanche, si l'on augmente la période de τ_1 de 4 à 5, et pour la même stratégie (global DM) le système résultant n'est pas ordonnançable : τ_3 rate son échéance à l'instant 8.



Conséquences des anomalies d'ordonnancement I

- Pour vérifier l'ordonnancement de tâches sporadiques il n'est d'aucune utilité d'étudier l'ordonnançabilité de l'ensemble périodique (et à départ simultané) correspondant.
- En d'autres termes, le cas périodique ne correspond pas au pire cas en multiprocesseur.
- À ce jour, on ne connaît pas le pire cas pour l'étude de tâches sporadiques.

Conséquences des anomalies d'ordonnancement II

- Les solutions aux problèmes d'ordonnancement multiprocesseur ne sont pas de triviales extensions des solutions monoprocesseurs
- Il faut être extrêmement prudent et rigoureux

Prédictibilité I

- Nous venons de constater qu'à propos de l'ordonnancement de tâches sporadiques nous ne disposons pas de la notion de pire cas.
- En revanche, pour l'ordonnancement de tâches périodiques/sporadiques les méthodes sont prédictibles, mais commençons par des définitions.

Définition (Prédictible)

Considérons deux ensembles de travaux J et J' qui diffèrent uniquement dans les durées d'exécution des travaux, avec les durées des travaux J' inférieures ou égales à la durée du travail correspondant dans J. Un algorithme d'ordonnancement est prédictible si lorsqu'il ordonnance J et J' (de manière séparée), le temps de terminaison de chaque travail de J' est toujours plus petit ou égal au temps de terminaison du travail correspondant de J.

Prédictibilité II

- Le caractère prédictible des algorithmes d'ordonnancement est important car il permet de vérifier l'ordonnançabilité en se focalisant sur les pire temps d'exécution des travaux ou tâches et non pas sur les temps d'exécution exacts.
- HA et LIU en 1994 ont montré que les ordonnanceurs qui attribuent des priorités fixes aux travaux (ou aux tâches) sont prédictibles.

Systèmes périodiques à échéance sur requête – Une condition nécessaire et suffisante I

- Remarquons que l'on a aucun espoir d'ordonnancer des systèmes avec
  - $U(\tau) > m$ (on demande trop de ressources) ou
  - $U_{\max}(\tau) > 1$ (le parallélisme au sein des travaux étant interdit)
- Par ailleurs, cette double condition est aussi suffisante pour les systèmes à échéance sur requête :

Théorème

Pour un ensemble périodique à échéance sur requête la condition suivante est suffisante et nécessaire pour l'ordonnançabilité sur m processeurs :

$$U(\tau) \leq m \qquad et \qquad U_{\max}(\tau) \leq 1$$

Systèmes périodiques à échéance sur requête – Une condition nécessaire et suffisante II

- Pour se convaincre de cette propriété nous pouvons songer à l'ordonnancement « processeur partagé », version multiprocesseur, dans lequel chaque tâche de τ_i reçoit la fraction $U(\tau_i)$ du temps d'un des processeurs entre son arrivée et sa terminaison. Dès lors, toutes les instances de τ_i seront ordonnançables.

EDF et systèmes à échéance sur requête

Théorème (Srinivasann 2002)

Pour un ensemble sporadique à échéance sur requête la condition suivante est suffisante pour l'ordonnançabilité sur m processeurs avec « global EDF *» :*

$$U(\tau) \leq m - (m-1)U_{\max}(\tau)$$

ou encore

$$m \geq \frac{U(\tau) - U_{\max}(\tau)}{1 - U_{\max}(\tau)}$$

L'algorithme EDF$^{(k)}$ I

- Afin d'alléger les notations nous allons supposer pour l'algorithme EDF$^{(k)}$ que $U(\tau_1) \geq U(\tau_2) \geq \cdots \geq U(\tau_n)$.
- Nous introduisons aussi la notation $\tau^{(i)}$ pour dénoter le système constitué des $(n - i + 1)$ tâches avec les plus petites utilisations de τ :

$$\tau^{(i)} \stackrel{\text{def}}{=} \{\tau_i, \tau_{i+1}, \ldots, \tau_n\}$$

- Remarquons dès lors qu'à partir du Théorème précédent nous pouvons déduire que

$$m \geq \lceil \frac{U(\tau) - U(\tau_1)}{1 - U(\tau_1)} \rceil$$

- Si l'on n'est pas dans l'obligation d'utiliser le « véritable » EDF on peut, dans certain cas, utiliser moins que $\lceil \frac{U(\tau)-U(\tau_1)}{1-U(\tau_1)} \rceil$ processeurs. Par exemple si l'on considère l'algorithme EDF$^{(k)}$

L'algorithme EDF$^{(k)}$ II

Définition (EDF$^{(k)}$)

- *Pour tout $i < k$, les travaux de τ_i ont une priorité maximale (ce qui peut se faire pour un ordonnanceur* EDF *en ajustant les échéances à $-\infty$).*
- *Pour tout $i \geq k$, les travaux de τ_i reçoivent une priorité définie par* EDF*.*

L'algorithme EDF$^{(k)}$ III

- En d'autres termes, l'algorithme EDF$^{(k)}$ donne une plus grande priorité aux travaux des $(k-1)$ premières tâches de τ et ordonnance les autres travaux en utilisant EDF.
- Notons que le « véritable » EDF correspond à EDF$^{(1)}$.

L'algorithme EDF$^{(k)}$ IV

Théorème (Goossens 2003)

Un système sporadique à échéance sur requête τ est ordonnancable sur m processeurs par l'algorithme EDF$^{(k)}$*, avec*

$$m = (k-1) + \lceil \frac{U(\tau^{(k+1)})}{1 - U(\tau_k)} \rceil$$

- Intuitivement, EDF$^{(k)}$ réserve un processeur pour les travaux de τ_1, un processeur pour les travaux de τ_2, ..., un processeur pour les travaux de τ_{k-1} et utilise EDF pour les travaux restants (i.e., $\tau_k, \ldots, \tau_n$) sur les processeurs restants.

L'algorithme EDF$^{(k)}$ V

Corollaire

Un système sporadique à échéance sur requête τ est ordonnancable sur $m_{\min}$ processeurs par l'algorithme EDF$^{(m_{\min})}$ *avec*

$$m_{\min}(\tau) \stackrel{\text{def}}{=} \min_{k=1}^{n} \left\{ (k-1) + \lceil \frac{U(\tau^{(k+1)})}{1 - U(\tau_k)} \rceil \right\} \quad (1)$$

Soit $k_{\min}(\tau)$ qui dénote le plus petit k qui minimise le membre de droite de l'équation 1 :

$$m_{\min}(\tau) = (k_{\min}(\tau) - 1) + \lceil \frac{U(\tau^{(k_{\min}(\tau)+1)})}{1 - U(\tau_{k_{\min}})} \rceil$$

L'algorithme $\text{EDF}^{(k)}$ – Example I

Considérons le système τ composé de 5 tâches :

	C	T
τ_1	9	10
τ_2	14	19
τ_3	1	3
τ_4	2	7
τ_5	1	5

$U(\tau) \approx 2.457$
$U_{\max}(\tau) = 0.9$

Nous pouvons vérifier pour ce système, que l'équation 1 est minimisée pour $k = 3$; dès lors, $k_{\min}(\tau) = 3$ et $m_{\min}(\tau)$ vaut 3
Donc, τ peut être ordonnancé avec l'algorithme $\text{EDF}^{(3)}$ sur 3 processeurs.

L'algorithme EDF$^{(k)}$ – Example II

- En revanche, le pour le « véritable » EDF on ne peut garantir que l'ordonnançabilité du système mais seulement si l'on dispose de $\lceil \frac{U(\tau)-U(\tau_1)}{1-U(\tau_1)} \rceil \approx \lceil 1.557/0.1 \rceil = 16$ processeurs.
- Des études expérimentales montrent que cette amélioration n'est pas anecdotique.

Systèmes périodiques à priorité fixe au niveau des tâches

Définition (Intervalle d'étude)

Pour un algorithme d'ordonnancement et un ensemble de tâches, un intervalle d'étude est un intervalle fini tel que le système est ordonnançable si et seulement si toutes les échéances dans l'intervalle d'étude sont respectées.

- Les premiers intervalles d'études sont basés sur des propriétés de périodicité des ordonnancements

Définitions et propriété

Définition (Ordonnanceur déterministe)

Un ordonnanceur est dit déterministe s'il produit un ordonnancement unique pour un ensemble de travaux donné.

Définition (Ordonnanceur sans mémoire)

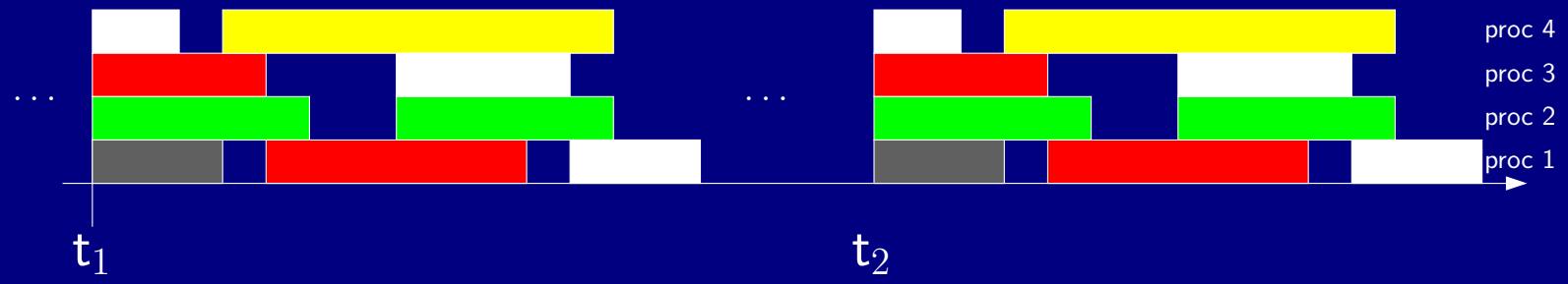
Un ordonnanceur est dit sans mémoire si ses décisions ne dépendent pas des décisions passées mais bien des caractéristiques des travaux actifs au moment de la décision.

Théorème (Cucu & Goossens 2006)

Soit $\mathcal{A}$ un ordonnanceur déterministe et sans mémoire, si le système périodique τ est ordonnançable sous l'égide de l'algorithme $\mathcal{A}$ alors l'ordonnancement est lui-même périodique (i.e., se répète à partir d'un instant).

Exploiter cette propriété de périodicité

- Nous savons que t_1 et t_2 existent tel que l'ordonnancement lui-même se répète dès t_2
- nous pouvons nous limiter à considérer l'intervalle fini $[0, t_2)$
- nous allons à présent étudier, pour différents modèles de tâches et ordonnanceurs, quand commence la répétition.



Système à départ simultané

Théorème (Cucu & Goossens 2006)

Soit $\mathcal{A}$ un ordonnanceur déterministe et sans mémoire, si le système périodique à départ simultané τ est ordonnançable sous l'égide de l'algorithme $\mathcal{A}$ alors l'ordonnancement est lui-même périodique dès l'instant 0.

Système à différé simultané et à échéance contrainte

Théorème (Cucu & Goossens 2006)

Soit $\mathcal{A}$ un ordonnanceur à priorité fixe au niveau des tâches et soit τ un système de n tâches périodiques à départ différé et à échéance contrainte, si τ est ordonnançable sous l'égide de l'algorithme $\mathcal{A}$ alors l'ordonnancement est lui-même périodique dès l'instant S_n avec la définition suivante :

$$S_i \stackrel{\text{def}}{=} \begin{cases} O_1 & \text{si } i = 1 \\ \max\{O_i, O_i + \lceil \frac{S_{i-1} - O_i}{T_i} \rceil T_i\} & \text{sinon} \end{cases}$$

et $P_i \stackrel{\text{def}}{=} \text{ppcm}\{T_1, T_2, \ldots, T_i\}$.

Système à différé simultané et à échéance arbitraire I

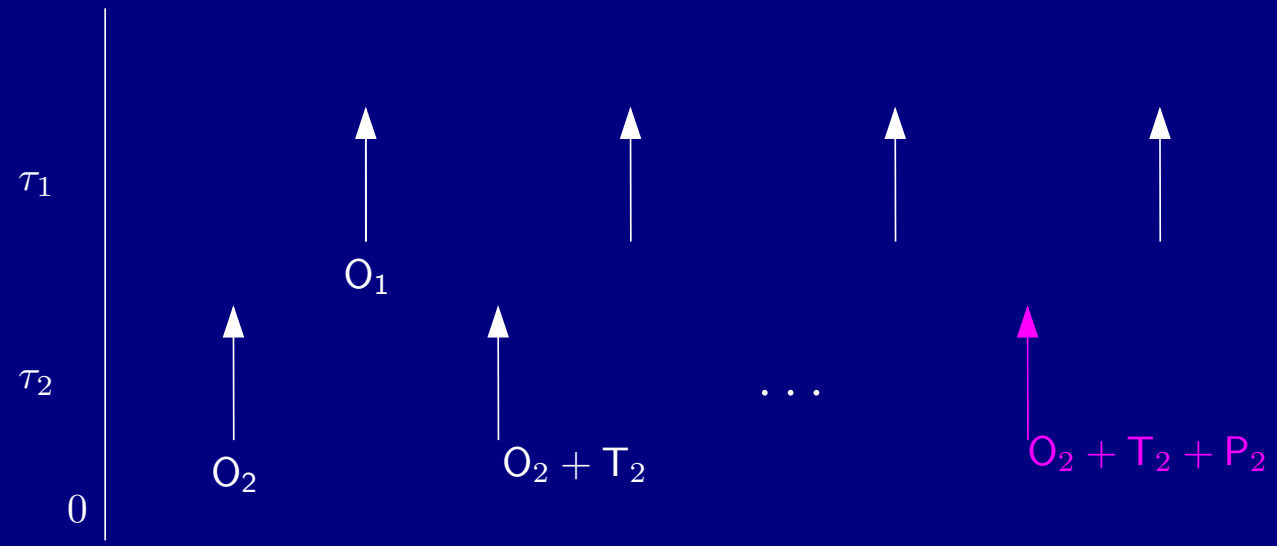
Théorème (Cucu & Goossens 2007)

Soit $\mathcal{A}$ un ordonnanceur à priorité fixe au niveau des tâches et soit τ un système de n tâches périodiques à départ différé et à échéance arbitraire, si τ est ordonnançable sous l'égide de l'algorithme $\mathcal{A}$ alors l'ordonnancement est lui-même périodique dès l'instant X_n avec la définition suivante :

$$X_i \stackrel{\text{def}}{=} \begin{cases} O_1 & \text{si } i = 1 \\ \max\{O_i, O_i + \lceil \frac{X_{i-1} - O_i}{T_i} \rceil T_i\} + P_i & \text{sinon} \end{cases}$$

et $P_i \stackrel{\text{def}}{=} \text{ppcm}\{T_1, T_2, \ldots, T_i\}$.

Système à différé simultané et à échéance arbitraire II



Tests d'ordonnançabilité I

Test

Un système périodique à départ simultané et à échéance arbitraire pour un ordonnanceur déterministe et sans mémoire est ordonnançable si et seulement si :

- *Toutes les échéances sont rencontrées dans l'intervalle* $[0, P_n)$
- *et le système à l'instant* P_n *est dans le même état qu'à l'instant* 0*, i.e., il y a exactement un travail actif pour chaque tâche à l'instant* P_n*.*

Tests d'ordonnançabilité II

Test

Un système périodique à départ différé et à échéance contrainte pour un ordonnanceur à priorité fixe au niveau des tâches est ordonnançable si et seulement si :

- *Toutes les échéances sont rencontrées dans l'intervalle* $[0, S_n)$
- *et le système à l'instant* $S_n + P_n$ *est dans le même état qu'à l'instant* S_n.

Tests d'ordonnançabilité III

Test

Un système périodique à départ différé et à échéance contrainte pour un ordonnanceur à priorité fixe au niveau des tâches est ordonnançable si et seulement si :

- *Toutes les échéances sont rencontrées dans l'intervalle* $[0, X_n)$
- *et le système à l'instant* $X_n + P_n$ *est dans le même état qu'à l'instant* X_n.

Introduction aux stratégies PFAIR I

- Nous allons présenter brièvement la famille d'algorithmes PFAIR. Nous nous limiterons à la définition de cette famille et de ses propriétés importantes.
- Les algorithmes PFAIR diffèrent fortement des algorithmes classiques dans la mesure où il est explicitement requis d'exécuter les tâches à un taux régulier (quasi constant).
- Dans les ordonnancements de tâches périodiques, chaque tâche τ_i s'exécute approximativement avec un taux de $U(\tau_i)$, si du moins l'on considère de grands intervalles. Cependant, sur de petits intervalles, le taux d'exécution de τ_i peut varier de manière importante.
- Dans les ordonnancements PFAIR, chaque tâche est exécutée quasiment à un taux constant en divisant la tâche en une série de sous-tâches.

Introduction aux stratégies PFAIR II

- Afin d'assurer le taux d'exécution de la tâche, les sous-tâches doivent s'exécuter dans des intervalles de tailles identiques appelés des fenêtres.
- Différentes sous-tâches d'une tâche peuvent s'exécuter sur des processeurs différents (la migration est donc autorisée) mais pas simultanément (c'est-à-dire que le parallélisme au sein d'une tâche est interdit).
- Dans le cas particulier de l'ordonnancement multiprocesseur identique et pour des tâches (strictement) périodiques à échéance sur requête, PFAIR est une stratégie optimale.

Ordonnancement *fair*

- Nous allons à présent caractériser de manière plus précise les ordonnancements PFAIR, nous considérons l'ordonnancement de tâches périodiques, à échéance sur requête et à départ simultané. Nous supposons que l'allocation des processeurs est discrète.
- Formalisons à présent la notion d'ordonnancement, il s'agit d'une fonction $\mathcal{S} : \tau \times \mathbb{Z} \mapsto \{0, 1\}$, où τ est l'ensemble de tâches périodiques. Lorsque $\mathcal{S}(\tau_i, t) = 1$ cela signifie que τ_i est ordonnancé dans l'intervalle $[t, t + 1)$ et $\mathcal{S}(\tau_i, t) = 0$ sinon.
- Dans un ordonnancement (idéal) parfaitement « *fair* », chaque tâche τ_i doit recevoir exactement $U(\tau_i) \times t$ unités de processeur dans l'intervalle $[0, t)$ (ce qui implique que toutes les échéances sont satisfaites). Cependant, un tel ordonnancement n'est pas possible dans le cas discret.

La notion de retard (*lag*)

- En revanche, PFAIR tente à chaque instant de répliquer le plus fidèlement possible cet ordonnancement idéal. La différence entre l'ordonnancement idéal et l'ordonnancement construit est formalisée par la notion de retard (*lag* dans la littérature d'origine). Plus formellement, le retard d'une tâche τ_i à l'instant t, dénoté par retard(τ_i, t) est défini de la manière suivante :

$$\text{retard}(\tau_i, t) \stackrel{\text{def}}{=} U(\tau_i) \cdot t - \sum_{\ell=0}^{t-1} \mathcal{S}(\tau_i, \ell). \tag{2}$$

Ordonnancement PFAIR I

Définition (Ordonnancement PFAIR)

Un ordonnancement est dit PFAIR *si et seulement si :*

$$-1 < \text{retard}(\tau_i, t) < 1 \qquad \forall \tau_i \in \tau, t \in \mathbb{Z}. \tag{3}$$

- De manière informelle, l'équation 3 demande que l'erreur d'allocation pour chaque tâche soit inférieure à une unité et ceci à chaque instant, ce qui implique que τ_i doit avoir reçu soit $\lfloor U(\tau_i) \cdot t \rfloor$, soit $\lceil U(\tau_i) \cdot t \rceil$.

Ordonnancement PFAIR II

Théorème

La contrainte de « Pfairness » implique nécessairement le respect des échéances.

Preuve : Une tâche périodique τ_i doit en effet recevoir C_i unités de processeur dans chaque intervalle $[\ell \cdot T_i, (\ell + 1) \cdot T_i)$, avec $\ell \geq 0$. À l'instant $t = \ell \cdot T_i$, $U(\tau_i) \cdot t = (C_i / T_i) \cdot \ell \cdot T_i = \ell \cdot C_i$, qui est un entier. Par l'équation 3, à l'instant $t = \ell \cdot T_i$ l'allocation de la tâche dans un ordonnancement PFAIR correspond à l'ordonnancement idéal. Puisque toutes les échéances sont satisfaites dans l'ordonnancement idéal elles le sont aussi dans l'ordonnancement PFAIR. ■

Optimalité de PFAIR

Le théorème suivant établi l'optimalité de PFAIR, et remarquons que ce résultat n'est pas contradictoire avec le Théorème 5 (slide 19) puisque nous considérons ici des tâches périodiques, dès lors l'ordonnanceur connaît parfaitement le futur.

Théorème (Baruah 96)

Soit τ un système périodique à échéance sur requête et départ simultané, il existe un ordonnancement PFAIR *sur m processeurs identiques de capacité 1 si et seulement si :*

$$U(\tau) \leq m \qquad et \qquad U_{\max} \leq 1. \tag{4}$$

Ordonnanceur(s) PFAIR

En ce qui concerne l'ordonnançabilité, à ce jour, trois algorithmes d'ordonnancement PFAIR optimaux ont été proposés : PF, PDet PD^2. Ces algorithmes donnent des priorités aux sous-tâches en utilisant une stratégie proche d'EDF, ils différent dans la manière de résoudre les cas d'égalités de priorité, c'est-à-dire les cas d'ambiguïtés.

Conclusions

- Introduction aux problèmes d'ordonnancement multiprocesseur ;
- Solutions ne sont pas de triviales extensions des solutions monoprocesseurs ;
- Discipline jeune.