Panorama restreint des méthodes d'ordonnancement temps réel

Antoine Bertout

22/09/21

LIAS, Université de Poitiers, ISAE-ENSMA

Plan

Introduction

Les objectifs

- Présenter un panorama rapide des résultats importants sur modèles simples (où l'on a le plus de résultats) du mono au multi.
- Essayer de distiller les notions à travers la présentations pour éviter l'effet catalogue.

Présentation en français *(english terms)*

Systèmes Temps-réel

Un système temps réel

- Fonctionnellement correct
- Temporellement correct : respect des contraintes de temps, des échéances

- Temps-réel **dur** (*hard*)
  - le non-respect d'une contrainte temporelle (échéance) est grave et peut avoir des conséquences catastrophiques (système de contrôle de vol)
- Temps-réel **mou** (*soft*)
  - certaines échéances de tâches peuvent être manquées sans danger
  - mais le nombre d'échéances manquées dégrade la qualité des résultats et doit être contrôlé

Dans la suite, nous nous concentrerons sur le temps réel dur.

Modélisation d’un système temps réel

Modélisation d'un système temps réel

On parle d'un système, qui comprend

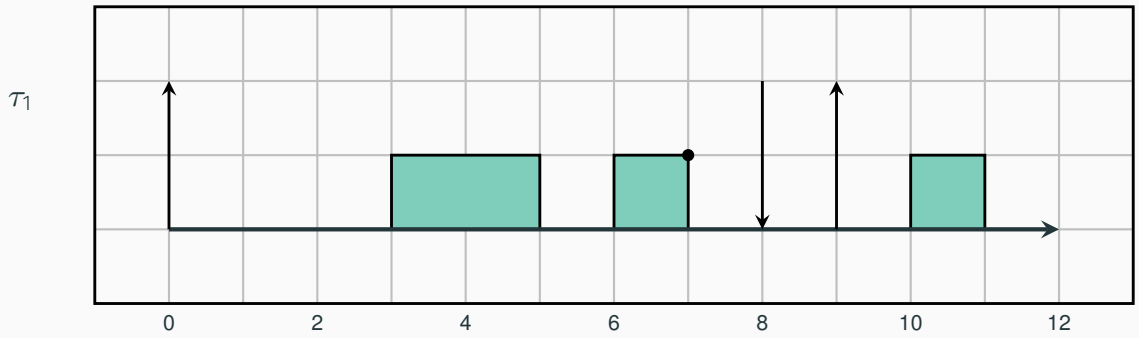
- L'application : ensemble de n tâches *(tasks)* implémentant les fonctionnalités du système
- Une plateforme : m processeur(s)

Modèle de tâche sporadique

- Une tâche τ_i génère un nombre infini d'instances de tâches également nommées travaux (*jobs*).

Une **tâche sporadique** τ_i est caractérisée par le tuple (C_i, D_i, T_i) avec

- C_i : temps d'exécution pire-cas d'un travail ou *WCET*. Borne supérieure sur le temps d'exécution
- T_i : durée minimale entre deux arrivées successives de travaux de la tâche
- D_i : échéance relative (*deadline*), délai maximal d'exécution autorisé après l'arrivée d'un travail



- Dates de réveil (*offset*) inconnues pour le sporadique et **synchrones** pour le périodique (premier travail d'une tâche relâché à $t = 0$))
- Tâches indépendantes

Modèle de tâche sporadique

Avertissement

Ce modèle de tâches indépendantes n'est pas représentatif des applications **réelles** mais il a permis de dégager des résultats fondamentaux sur des problèmes praticables. Il a aussi rendu possible la définition de nombreux concepts utiles pour raisonner sur des modélisations plus réalistes d'application.

NB : Sauf mention contraire, les résultats qui vont suivre correspondent au modèle de tâches sporadiques.

Hors Programme

Ce dont je ne parlerai malheureusement donc pas 😕

- tâches dépendantes (précédences, DAG)
- ressources partagées (blocage, héritages de priorité, etc.)
- tâches parallèles
- criticité mixte
- tâches à suspension
- modèles probabilistes
- etc.

Modèle de tâche sporadique

Schéma d'activation

- Si T_i correspond à la durée **exacte** et non **minimale** qui sépare l'arrivée de deux instances $\Rightarrow$ on parle de tâches périodiques. Le cas périodique est donc un cas particulier du sporadique.

Types d'échéances

- Implicite : $D_i = T_i$
- Contrainte : $D_i \leq T_i$
- Arbitraire : pas liée à T_i mais $\geq C_i$

- L'utilisation U d'un ensemble de tâches $\mathbf{T}$ est la somme de l'utilisation de ses tâches : $U = \sum_{\forall \tau_i \in \mathbf{T}} \frac{C_i}{T_i}$
- Notion de densité D pour les tâches à échéances non-implicites : $D = \sum_{\forall \tau_i \in \mathbf{T}} \frac{C_i}{min(D_i, T_i)}$

Conseil

Toujours bien poser le contexte et les hypothèses. ..

Terminologie de l’ordonnancement

Terminologie de l'ordonnancement

On dispose d'une plateforme d'exécution matérielle (avec un système d'exploitation) et une application de tâches récurrentes.

Quel travail exécuter et quand ?

$\Rightarrow$ Rôle de l'ordonnanceur, la partie du système d'exploitation qui décide quel travail exécuter.

L'ordonnanceur va suivre les règles d'un algorithme d'ordonnancement, qui peut être

- hors-ligne (statique) : ordonnancement fixé a priori, à la conception (tables).
- **en-ligne** : les décisions sont prises pendant l'exécution du système.

Terminologie de l'ordonnancement

On distingue plusieurs classes d'algorithmes à base de priorités.

- Priorité fixe par tâche (*fixed-task priority* ou **FTP**). La priorité est la même pour tous les travaux d'une tâche. Par exemple, DM.
- Priorité fixe par travail (*fixed-job priority* ou **FJP**). Par exemple, EDF.
- Priorité dynamique par travail (*dynamic priority* ou **DP**). La priorité d'un travail peut varier au cours du temps. Par exemple, LLF.

Terminologie de l'ordonnancement

Encore un peu de vocabulaire sur les algo. d'ordonnanncement.

Conservatif

- Un algo. d'ordonnancement est **conservatif** (*work-conserving*) s'il ne reste jamais oisif (*idle*) lorsque des travaux sont prêts à s'exécuter.

Préemptif/non-préemptif

- Un algo. d'ordonnancement est **préemptif** s'il peut choisir de suspendre un travail en cours d'exécution (non-terminé) pour en exécuter un autre.
- Il est **non-préemptif** s'il laisse toujours un travail en cours d'exécution se terminer avant de procéder à celle d'un autre.

Dans cette présentation, on considère les algorithmes préemptifs et on néglige les coûts de préemptions (confortable non ? 🙂)

Terminologie de l'ordonnancement

- Ordonnançabilité : Un ensemble de tâches est **ordonnançable** selon un algorithme d'ordonnancement et une plateforme si toutes les séquences valides de travaux qui peuvent être générées par l'ensemble de tâches peuvent être exécutées sans rater une échéance.
- Faisabilité : Un ensemble de tâches est **faisable** s'il existe un algorithme d'ordonnancement capable de l'ordonnancer (de le rendre ordonnançable [1]).
- Optimalité : un algorithme d'ordonnancement est **optimal** s'il peut ordonnancer n'importe quel ensemble de tâches faisable.

[1] ou qu'il l'ordonnance fiablement

Monoproc.

Monoproc. : Ordonnancement à priorité fixe par tâche

Dans la première partie de la présentation, nous plaçons dans le contexte monoprocesseur.

Principe de l'ordonnancement FTP

- Chaque tâche τ_i se voit attribuer une priorité.
- La tâche active avec la priorité la plus haute est exécutée.
- **(non-)convention** : parfois plus le nombre est grand plus la priorité est élevée, parfois l'inverse **(à bien définir donc)**.

Monoproc. : Ordo. FTP - Assignation de priorité

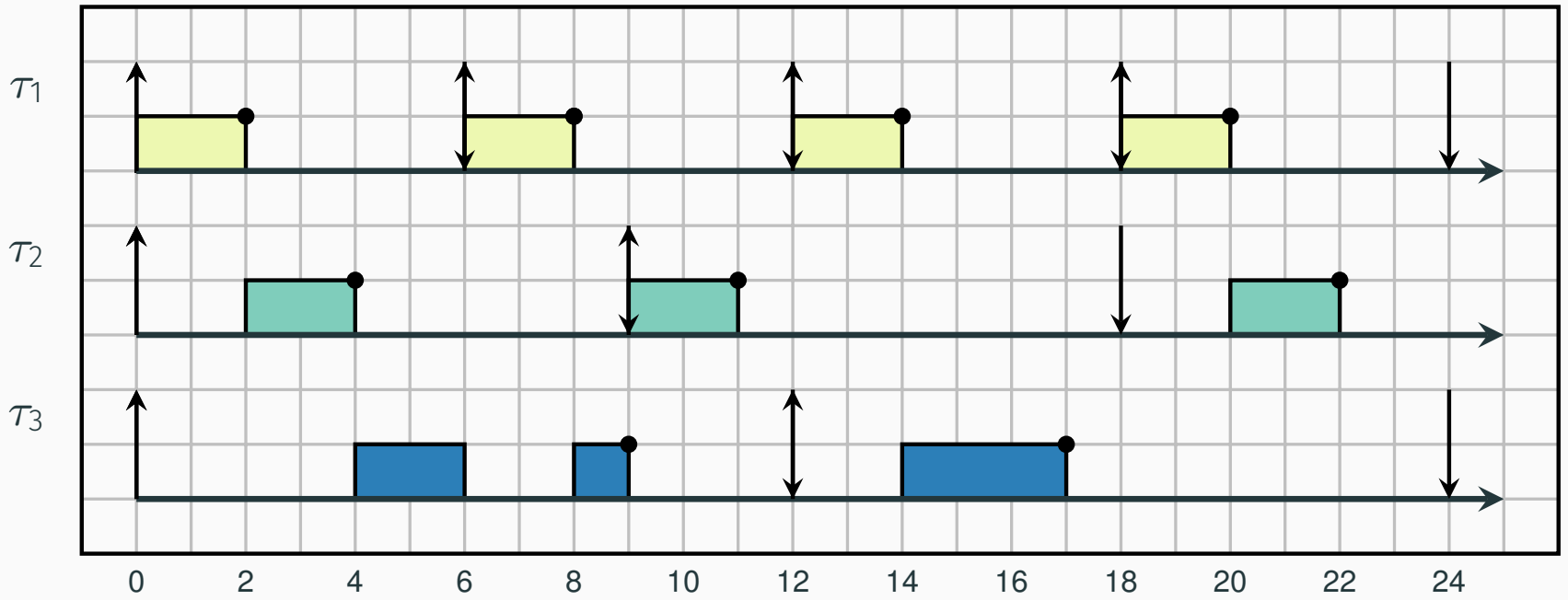
Comment assigner les priorités ? et pour quelles garanties ?

- On cherche l'assignation de priorités qui puisse rendre le système ordonnançable
- On cherche donc l'assignation optimale (en terme de respect des échéances) OPT de priorités.

Rappel

- Si une assignation de priorité rendant le système ordonnançable existe, alors OPT rendra le système ordonnançable.
- Si le système n'est pas ordonnançable selon OPT, alors aucune assignation de priorité rendant le système ordonnançable n'existe.

Exemple avec trois tâches $\tau_1(2, 6, 6)$,$\tau_2(2, 9, 9)$ et $\tau_3(3, 12, 12)$. τ_1 a la priorité la plus haute **HP**, τ_3 la plus basse **LP** et τ_2 est de priorité intermédiaire **MP**.



Résultats fondateurs

1973 Liu et Layland

- L'assignation de priorité Rate-Monotonic (RM) est optimale pour les tâches périodiques à échéances implicites.
- RM donne aux tâches de périodes les plus courtes les priorités les plus élevées.

1982 Leung et Whitehead

- L'assignation de priorité Deadline-Monotonic (DM) est optimale pour les tâches périodiques à échéances contraintes.
- DM donne aux tâches d'échéances les plus courtes les priorités les plus élevées.

Remarques

- DM n'est ni optimal pour le cas asynchrone, ni pour les échéances arbitraires $\Rightarrow$ Audsley (1991 et 2001)
- Ces assignations sont les meilleures également pour le cas sporadique, nous verrons pourquoi plus tard.

Monoproc. Vérification de l'ordonnançabilité

Hypothèse : nous connaissons une assignation optimale de priorité pour notre contexte (par ex. tâches périodiques, $D_i \leq T_i$).

Comment vérifier l'ordonnançabilité de mon ensemble de tâches ?

- Simulation : simulation de l'exécution du système et vérification qu'aucune échéance n'est manquée.
- Les maths : analyses/tests par l'application de formules.

Ces vérifications constituent des conditions d'ordonnançabilité.

- A est une condition nécessaire pour B, $B \Rightarrow A$
- A est une condition suffisante pour B, $A \Rightarrow B$
- A est une condition nécessaire et suffisante pour B, $A \Leftrightarrow B$

Questions

- Les WCET sont des pire-cas, une *vraie* exécution présentera généralement des temps plus courts, non ?
- Dans le cas sporadique, je ne connais que la séparation minimale entre deux activations successives, non ?
- Sur quel intervalle dois-je simuler pour m'assurer que la simulation est sûre ?

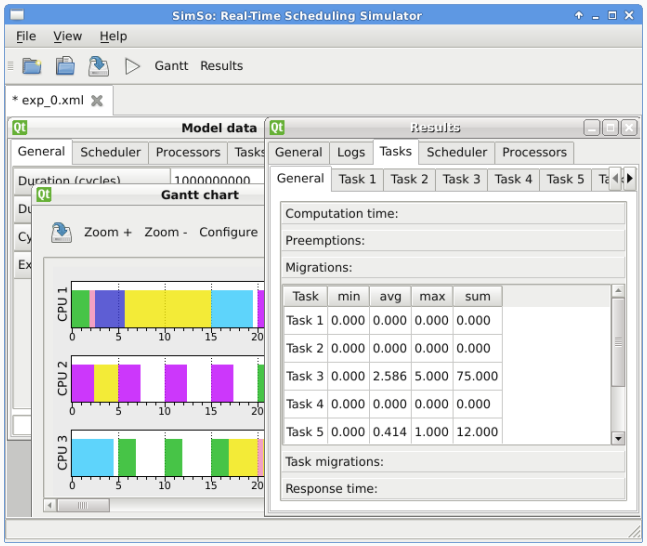


Fig. 1 : Simulateur *Simso [1]* (existe en version web)

Résultats majeurs en monoprocesseur (m = 1)

Réponses

Pour que la simulation soit sûre :

- L'algorithme d'ordonnancement doit être viable (*sustainable*) eu égard aux paramètres concernés.
- L'intervalle de temps simulé doit être un intervalle de faisabilité (ou d'étude).

Définition (*A*-Viabilité)

La A-viabilité formalise l'idée selon laquelle, un changement (possiblement en-ligne) intuitivement positif d'un paramètre A apporté à un système ordonnançable ne peut pas le rendre non-ordonnançable.

On parlera par exemple de *C-viabilité* pour les pire-temps d'exécution (aussi appelée *prédictibilité*).

NB : La définition s'applique également aux tests en général.

Résultats majeurs en monoprocesseur (m = 1) - Viabilité

Viabilité

Les algorithmes d'ordonnancement FJP et FTP préemptifs synchrones monoprocesseur pour des ensembles de tâches périodiques et sporadiques sont **C-viable**, **D-viable** et **T-viable**.

- Dans ce contexte, la simulation peut être considérée comme viable.
- le cas **périodique** implique l'ordonnançabilité d'un système **sporadique** (pire-cas probable).
- le cas **synchrone** implique l'ordonnançabilité d'un système **asynchrone** (ou non-concret, la réciproque est fausse).
- Les cas contre-intuitifs sont appelés anomalies. Nous en reparlerons dans le cadre du multiprocesseur.

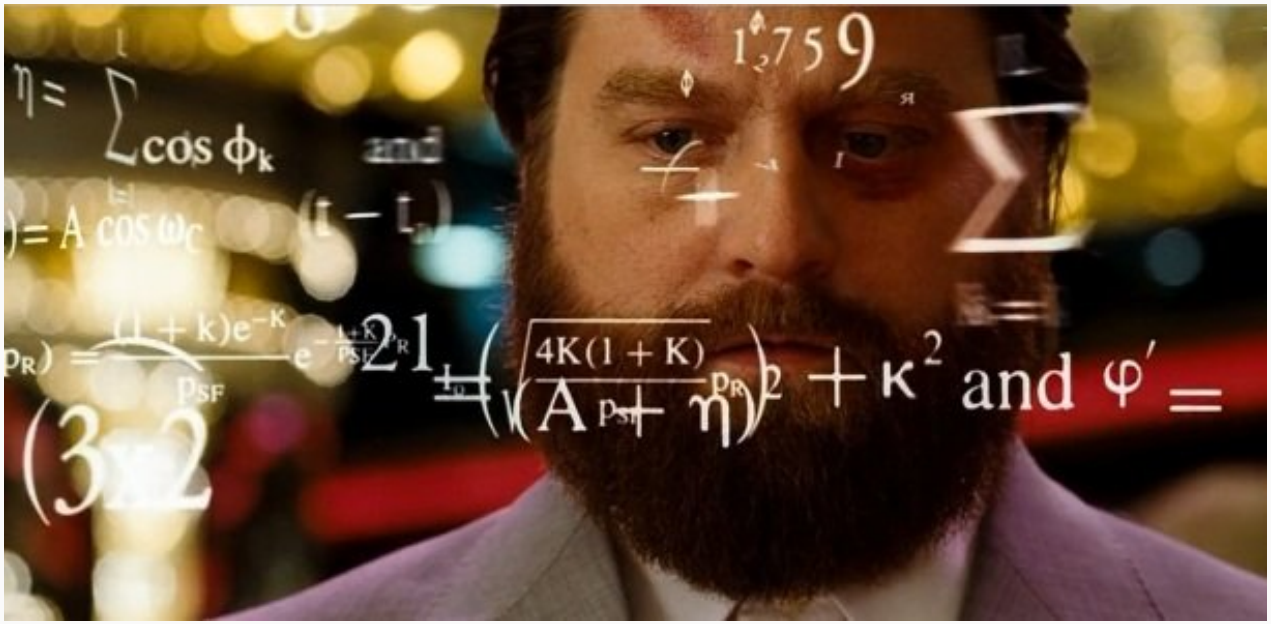
Intervalle de faisabilité

- $[0, H(\mathrm{T}))$ (dans notre contexte) où l'hyperpériode $H(\mathrm{T})$ est le PPCM des périodes de T...et ça peut vite monter.
- $T_1 = 10, T_2 = 15, T_3 = 30, PPCM(T_i) = 30$
- $T_1 = 9, T_2 = 16, T_3 = 29, PPCM(T_i) = 4176$
  $\Rightarrow$ La simulation est de complexité **exponentielle**.
- Sur un intervalle non sûr, la simulation peut faire office de **test nécessaire**.

Tests/Analyses d'ordonnançabilité

La majorité des analyses d'ordonnançabilité rentrent dans ces deux catégories :

- des conditions suffisantes
- des conditions nécessaires et suffisantes (**tests exacts**)



Condition suffisante

Définition (Borne (inférieure) d'utilisation)

Tous les systèmes ayant une utilisation inférieure ou égale à la borne d'utilisation U_A sont ordonnançables selon l'algorithme ou la classe d'algorithmes donnée A. ***Il s'agit donc d'une condition suffisante mais non nécessaire.***

Résultat de Liu et Layand 1973

Soit $\mathbf{T}$ un ensemble de de n tâches périodiques avec $D_i = T_i$ ordonnancé selon RM

- $\mathbf{T}$ ordonnançable si $U \leq U_{LL} = n(2^{1/n} - 1)$.
- U_{LL} tend vers $\ln(2) \approx 0.69$ quand $n \to \infty$ (fonction décroissante)

Test suffisant

Beaucoup d'ensembles ratent le test et ... sont ordonnançables.

Condition nécessaire et suffisante

Dans le cas FTP, on utilise généralement l'analyse de temps de réponse (RTA) [2] pour vérifier l'ordonnançabilité.

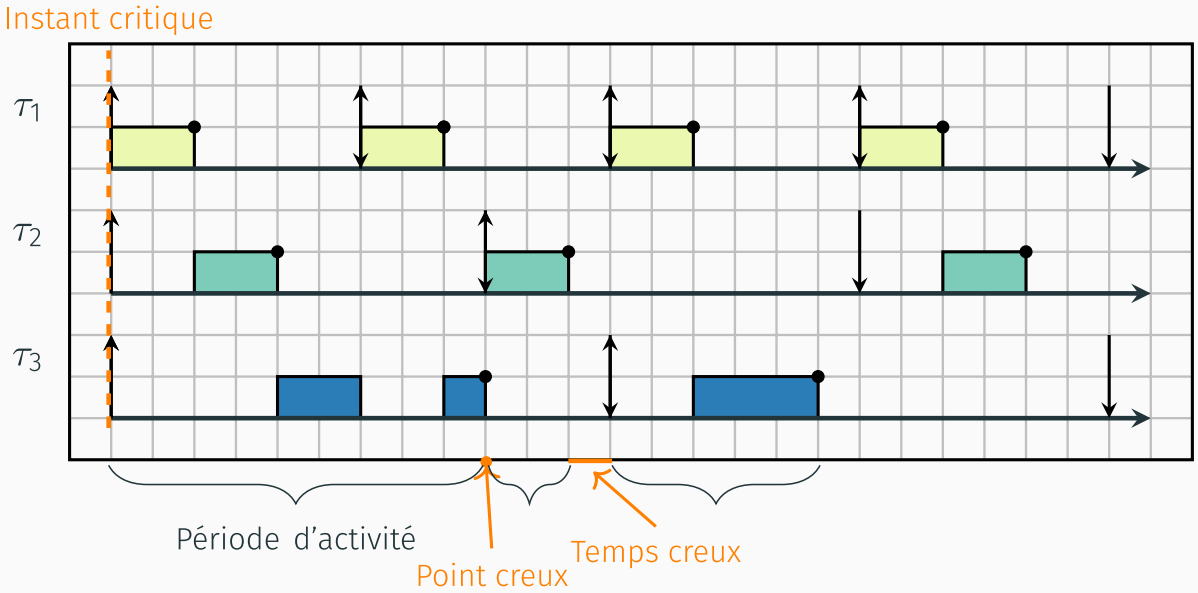
- Le temps de réponse pire-cas ($WCRT$) R_i est la durée la plus longue que peut prendre un travail entre son arrivée et la fin de son exécution.
- Système ordonnançable si $\forall \tau_i, R_i \leq D_i$.
- L'analyse du temps réponse constitue un **test exact** de complexité **pseudo-polynomiale**.

Théorème (Liu et Layland 1973)

Le WCRT d'une tâche se produit lorsqu'elle est réveillée simultanément avec les tâches plus prioritaires qu'elle-même.

RTA se base sur les notions d'instant critique pour les tâches synchrones et de période d'activité.

- LE WCRT d'une tâche survient dans la plus longue période d'activité de son niveau de priorité
- La plus longue période d'activité est initiée par l'instant critique



$$R_i^{k+1} = C_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil \cdot C_j \qquad \text{avec } R_i^0 = C_i$$

- Temps de réponse R_i^k, initialisé à C_i
- Interférence : nombre d'activations $\left\lceil \frac{R_i}{T_j} \right\rceil$ des tâches de priorité égale ou supérieure sur R_i^k, $\times$ leurs WCET respectifs.
- Itération de point fixe
  - Arrêt si $R_i^k > D_i \Rightarrow$ non-ordonnançable
  - Sinon si $R_i^{k+1} = R_i^k \Rightarrow$ ordonnançable

$$R_3^0 = C_3$$

$$R_3^1 = 3 + \left\lceil \frac{3}{6} \right\rceil \cdot 2 + \left\lceil \frac{3}{9} \right\rceil \cdot 2 = 7$$

$$R_3^2 = 3 + \left\lceil \frac{7}{6} \right\rceil \cdot 2 + \left\lceil \frac{7}{9} \right\rceil \cdot 2 = 9$$

$$R_3^2 = 3 + \left\lceil \frac{9}{6} \right\rceil \cdot 2 + \left\lceil \frac{9}{9} \right\rceil \cdot 2 = 9$$

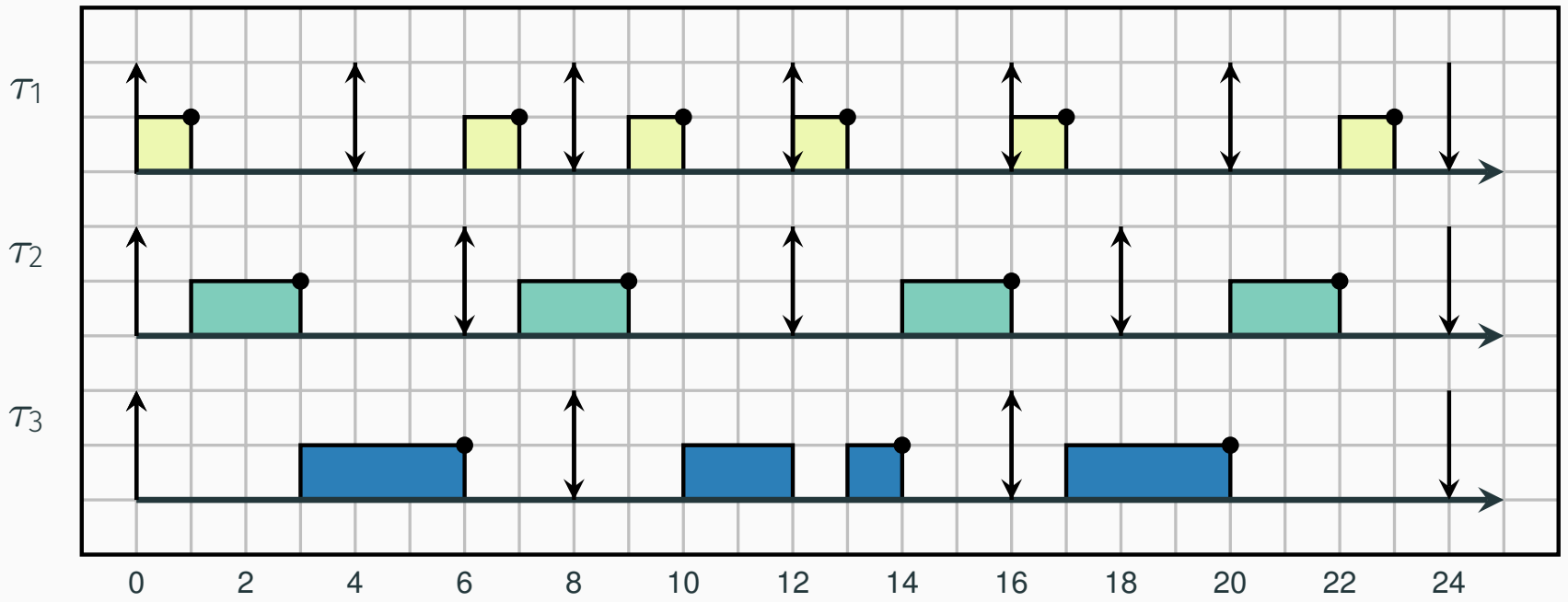
$$R_3 = 9 \leq D_3 \Rightarrow \tau_3 \text{ est ordonnançable}$$

Principe de l'ordonnancement FJP

- La priorité d'une tâche τ_i peut varier pour chaque travail.
- L'algorithme majeur FJP est Earliest Deadline First (EDF)
- Règles
  - Plus l'échéance absolue d'un travail est proche, plus sa priorité est élevée.
  - En cas d'ex æquo : on fait comme on veut !

Monoproc. : Ordonnancement à priorité fixe par travail- Exemple

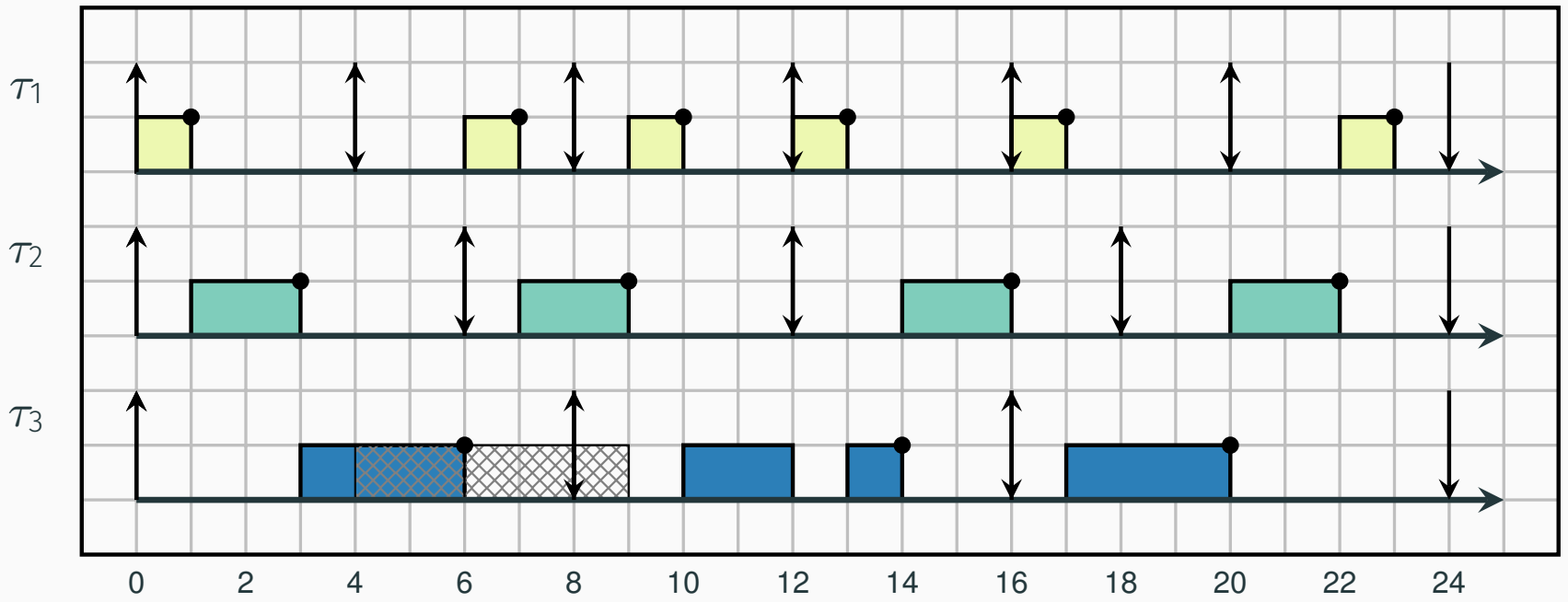
Exemple avec trois tâches $\tau_1(1,4,4)$,$\tau_2(2,6,6)$ et $\tau_3(3,8,8)$.



- ex æquo traités différemment à $t = 4$ et à $t = 12$.
- ce système n'est pas ordonnançable avec RM/DM (le premier travail de τ_3 finirait à $t = 10 > 8$)
- EDF (FJP) domine RM/DM (FTP).

Monoproc. : Ordonnancement à priorité fixe par travail- Exemple

Exemple avec trois tâches $\tau_1(1,4,4)$,$\tau_2(2,6,6)$ et $\tau_3(3,8,8)$.



- ex æquo traités différemment à $t = 4$ et à $t = 12$.
- ce système n'est pas ordonnançable avec RM/DM (le premier travail de τ_3 finirait à $t = 10 > 8$)
- EDF (FJP) domine RM/DM (FTP).

Comparaison des algorithmes d'ordonnancement

Soit deux algorithmes d'ordonnancement A et B :

- A domine B si A ordonnance tous les ensembles de tâches que B ordonnance mais qu'il existe des ensembles que A ordonnance mais pas B
- A et B sont équivalents A ordonnance tous les ensembles de tâches que B ordonnance et vice-versa
- A et B sont incomparables. Certains ensembles de tâches sont ordonnançables selon A mais pas B et vice-versa.

Ces définitions s'appliquent aussi pour les tests. Par exemple, en FTP, la borne hyperbolique [3] $\prod_{i=1}^{n}(u_i + 1) \leq 2$ **domine** la borne de Liu et Layland vue précédemment.

Théorème (Dertouzos 1974)

EDF est optimal en monoprocesseur pour les tâches sporadiques.

Vérifier l'ordonnançabilité sous EDF

Si $D_i = T_i$, la condition suivante est **nécessaire et suffisante** (exacte) pour un ensemble de tâches d'utilisation totale U :

$$U \leq 1$$

Dans le cas où $D_i \neq T_i$, on peut étudier la demand bound function (DBF).

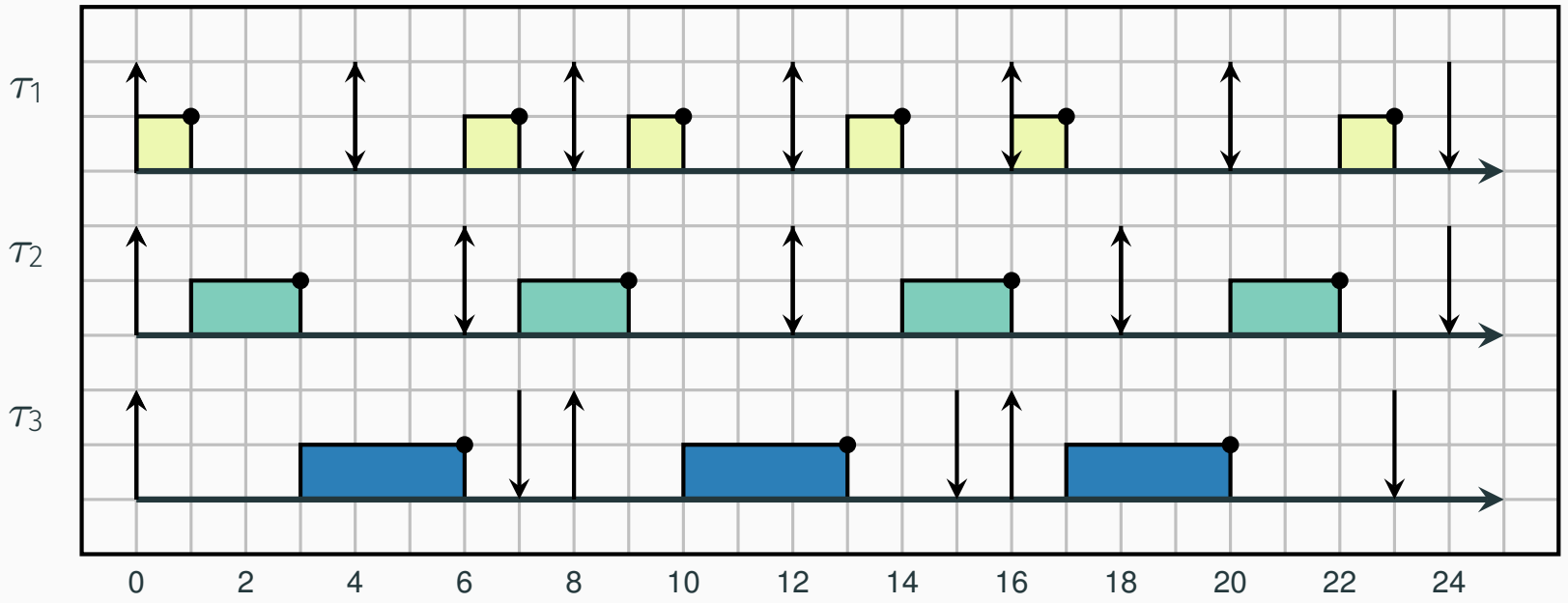
- La fonction de demande *dbf* calcule la quantité de travail qu'un processeur doit fournir dans un intervalle donné pour exécuter les tâches dont les échéances (et les arrivées nécessairement) s'y trouvent.
- Pour les tâches synchrones, la *dbf* sur tout intervalle $[t, t + L]$ est bornée par celle sur $[0, L]$ ce qui limite les intervalles à vérifier.
- La demande d'une tâche τ_i sur $[0, t]$ est

$$dbf_i(t) = \left(\left\lfloor \frac{t - D_i}{T_i} \right\rfloor + 1 \right)_0 \cdot C_i$$

- Système ordonnançable si et seulement si $\forall t \in [0, H], \sum_i dbf_i \leq t$. [2]
- Cette *dbf* constitue un **test exact** (cas synchrone et échéances contraintes) de complexité **pseudo-polynomiale**.

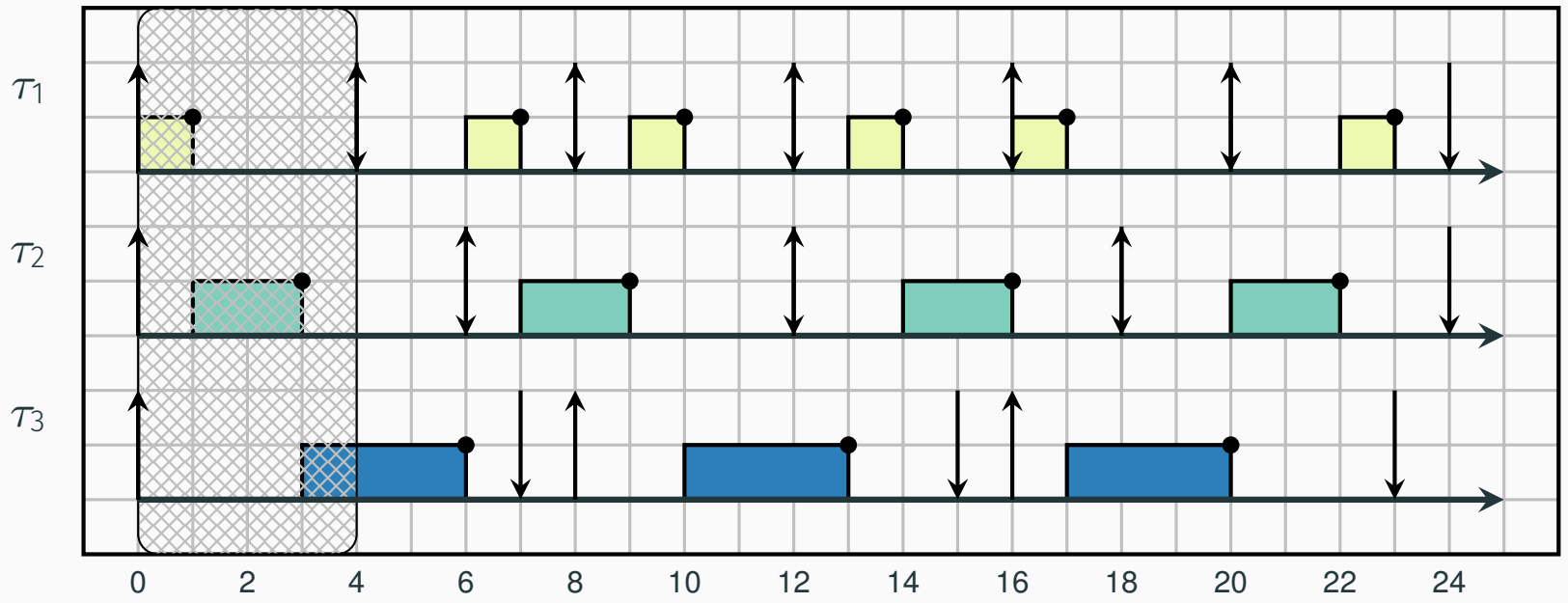
[2] En fait, il suffit de tester les valeurs de t correspondant à des échéances et on peut utiliser une borne supérieure de l'intervalle moins pessimiste.

Exemple avec $\tau_1(1, 4, 4)$,$\tau_2(2, 6, 6)$ et $\tau_3(3, 7, 8)$ à quelques dates t



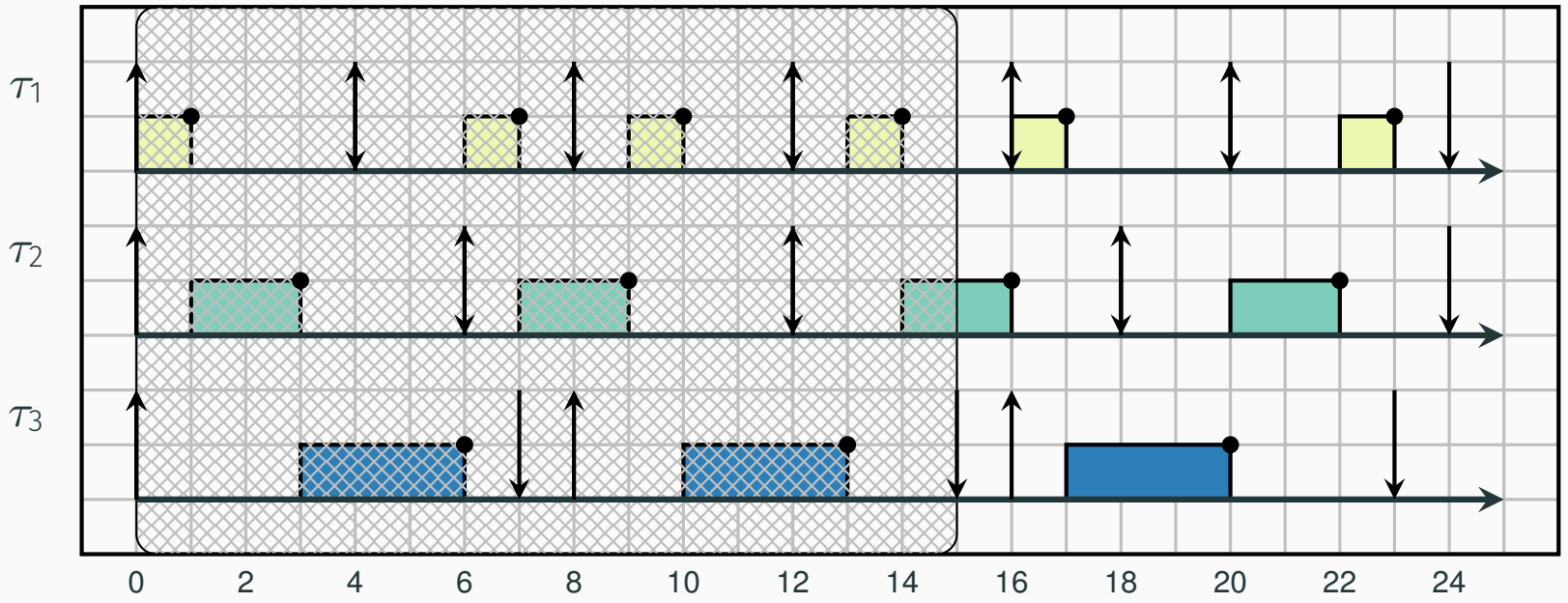
Monoproc. : Ordonnancement à priorité fixe par travail - DBF

Exemple avec $\tau_1(1,4,4)$,$\tau_2(2,6,6)$ et $\tau_3(3,7,8)$ à quelques dates t



$$dbf(4) = dbf_1(4) + dbf_2(4) + dbf_3(4) = 1 \cdot C_1 + 0 + 0 = 1 \leq 4$$

Exemple avec $\tau_1(1,4,4)$,$\tau_2(2,6,6)$ et $\tau_3(3,7,8)$ à quelques dates t

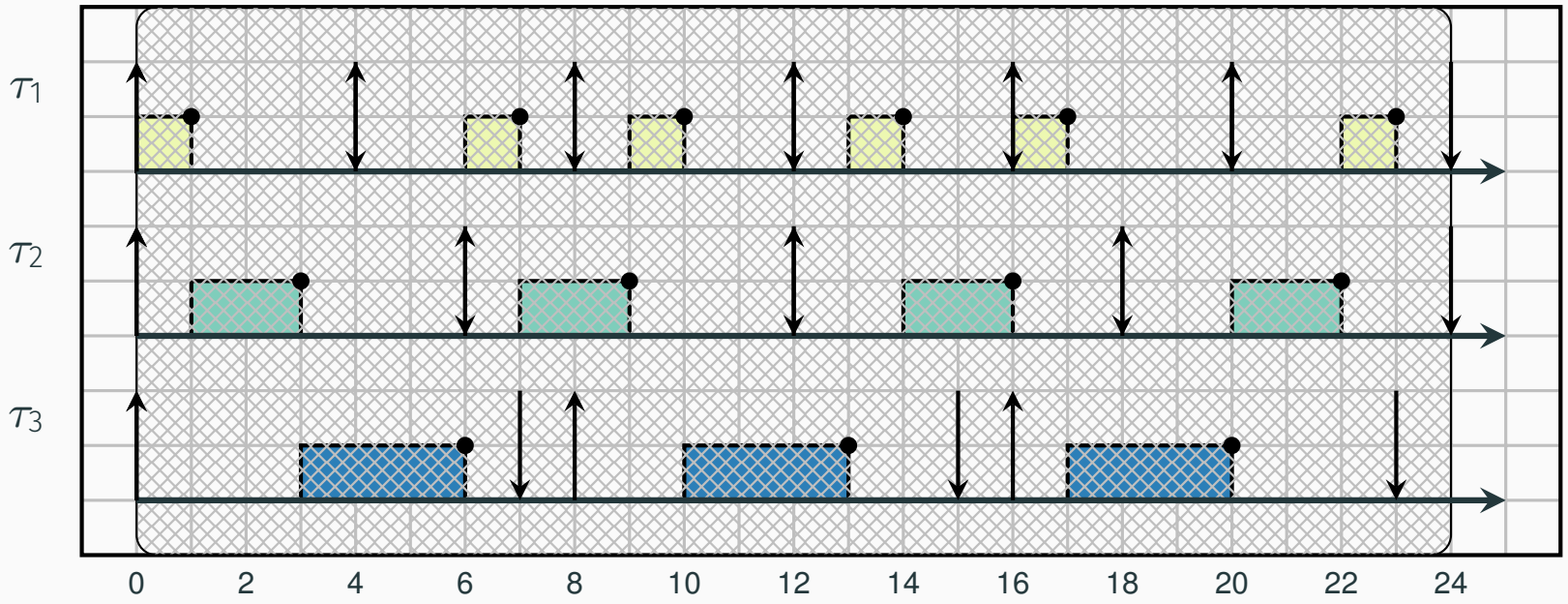


$dbf(4) = dbf_1(4) + dbf_2(4) + dbf_3(4) = 1 \cdot C_1 + 0 + 0 = 1 \leq 4$

...

$dbf(15) = dbf_1(15) + dbf_2(15) + dbf_3(15) = 3 \cdot C_1 + 2 \cdot C_2 + 2 \cdot C_3 = 13 \leq 15$

Exemple avec $\tau_1(1,4,4)$,$\tau_2(2,6,6)$ et $\tau_3(3,7,8)$ à quelques dates t



$dbf(4) = dbf_1(4) + dbf_2(4) + dbf_3(4) = 1 \cdot C_1 + 0 + 0 = 1 \leq 4$

...

$dbf(15) = dbf_1(15) + dbf_2(15) + dbf_3(15) = 3 \cdot C_1 + 2 \cdot C_2 + 2 \cdot C_3 = 13 \leq 15$

...

$dbf(24) = dbf_1(24) + dbf_2(24) + dbf_3(24) = 6 \cdot C_1 + 4 \cdot C_2 + 3 \cdot C_3 = 23 \leq 24$

Ordonnancement à priorité dynamique

Lorsque la priorité d'un travail peut varier au cours du temps, on parle d'ordonnancement à priorité dynamique (DP).

Least Laxity First fait partie de cette classe.

- Il assigne la priorité la plus grande au travail ayant la plus petite laxité.
- La laxité d'un travail est une indication de la marge qu'il lui reste pour terminer son exécution.
  Laxité = échéance absolue - (temps d'exécution restant + t)
- Si une tâche avec une laxité nulle à instant t n'est pas exécutée, elle va nécessairement manquer son échéance si elle atteint son WCET.

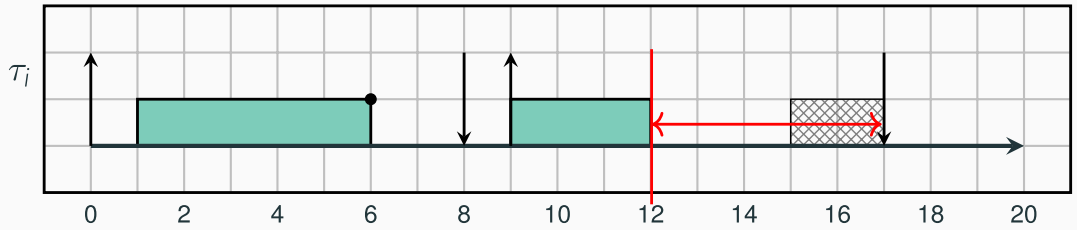


Fig. 2 : Laxité de 17-(12+2) = 3. A $t = 12$, il reste 2 unités de temps à exécuter pour τ_i et 5 disponibles avant son échéance.

- LLF est optimal pour les tâches sporadiques comme EDF,
- mais LLF induit plus de décisions d'ordonnancement

Intro. Multipro.

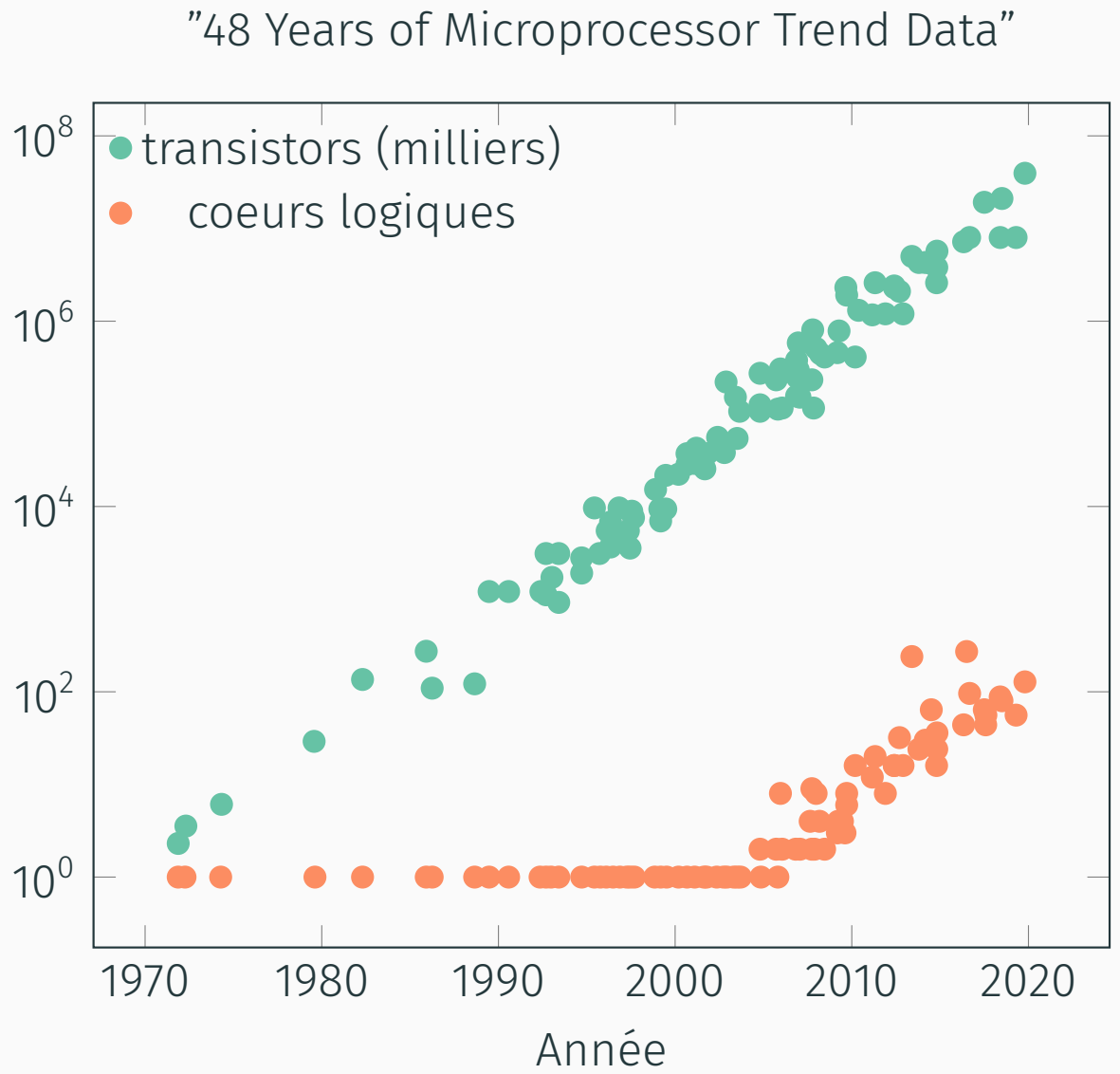


Fig. 3 : Données originales jusque 2010 collectée par M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten" et pour 2010-2019 par K. Rupp

La loi d'Amdhal caractérise l'accélération théorique S que l'on peut espérer en fonction de la portion du code parallélisé :

$$S = \frac{1}{1 - p + p/m}$$

où p représente le pourcentage de code parallélisé et m le nombre de coeurs identiques.

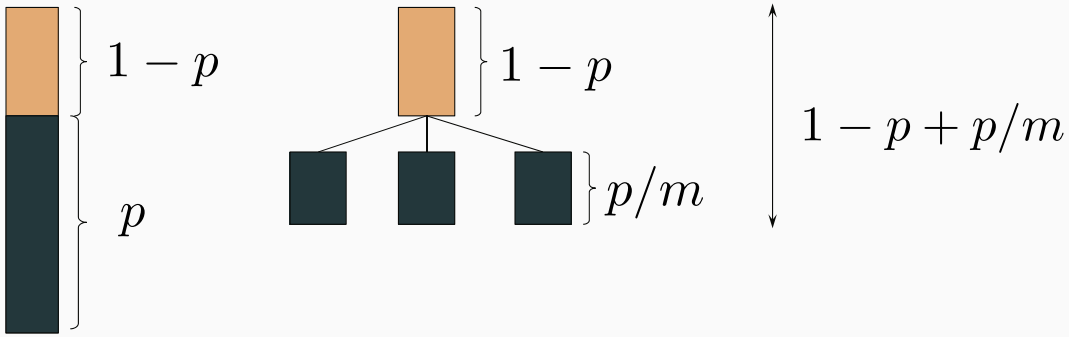


Fig. 4 : Représentation visuelle de la loi d'Amdhall. Repris de [4].

Quelle est l'accélération maximale à laquelle ont peut s'attendre en passant d'un processeur monocoeur à un processeur à 100 coeurs avec un code parallélisé à 50 % ?

La loi d'Amdhal caractérise l'accélération théorique S que l'on peut espérer en fonction de la portion du code parallélisé :

$$S = \frac{1}{1 - p + p/m}$$

où p représente le pourcentage de code parallélisé et m le nombre de coeurs identiques.

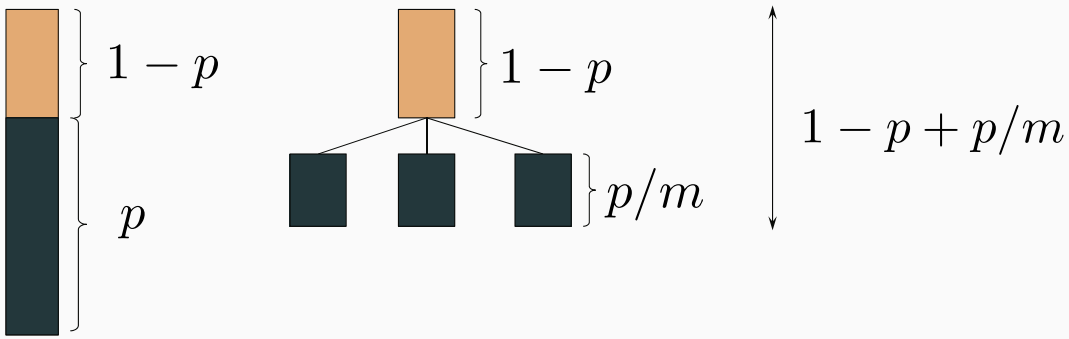


Fig. 4 : Représentation visuelle de la loi d'Amdhall. Repris de [4].

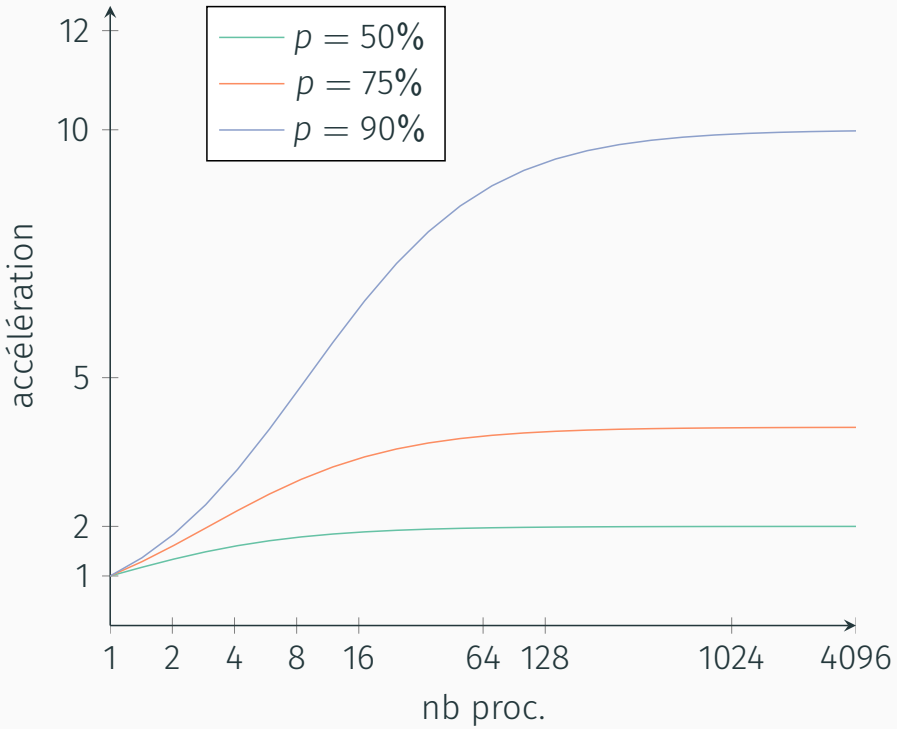
Quelle est l'accélération maximale à laquelle ont peut s'attendre en passant d'un processeur monocoeur à un processeur à 100 coeurs avec un code parallélisé à 50 % ? $S = \frac{1}{1-0.5+0.5/100} \approx 1.98$

Intro. Multipro. : Loi d'Amdhall

Y a-t-il un intérêt à multiplier encore le nombre de coeurs par 10 ? par 100 ?

Intro. Multipro. : Loi d'Amdhall

Y a-t-il un intérêt à multiplier encore le nombre de coeurs par 10 ? par 100 ?



- le speedup tend vers 2, quelle que soit l'augmentation du nb de coeurs
- rapidement c'est p qui limite l'accélération
- $\lim_{m\to\infty} S = \frac{1}{1-p}$ et pas d'intérêt ici à augmenter le nombre de coeurs si le code n'est pas plus parallélisé.

Dans la suite pas de parallélisation prise en compte dans l'ordonnancement

Ordonnancement multiprocesseur

En route vers le multiprocesseur

En monoprocesseur, les questions à se poser étaient principalement

- Quel(le) tâche/travail exécuter ?
- Quand ?

$\Rightarrow$ Le passage au multiprocesseur ajoute une dimension spatiale au problème.

Classification Multipro.

Classification Multipro. : Plateformes

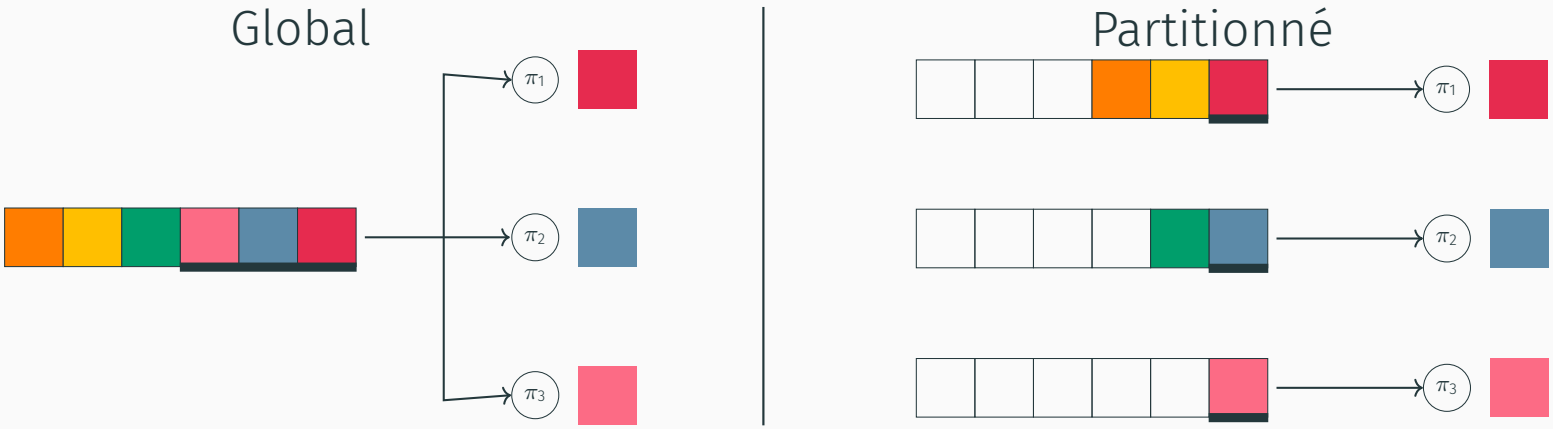
Différents types de plateforme :

- processeurs identiques : même vitesse pour tous les processeurs
- processeurs **uniformes** (related) : vitesse v attachée à un processeur pour toutes les tâches. *Exemple : deux processeurs de même architecture mais cadencés à fréquences différentes.*
- processeurs **hétérogènes** (unrelated) : vitesse v propre à chaque couple (tâche / processeur). *Exemple : SoC avec CPU, GPU, FPGA, etc.*

Classification Multipro. : Migrations

De manière extrême, on peut séparer les politiques d'ordonnancement multiprocesseur en deux catégories :

- Ordonnancement **partitionné**
- Ordonnancement **global**



Un peu plus finement :

Ordonnancement global

- avec migration limitée aux tâches (à migration restreinte) : les tâches sont autorisées à migrer, les travaux ne peuvent pas migrer.
- avec **migration totale** : les travaux peuvent migrer d'un processeur à l'autre.

Parallélisme intra-tâche

NB : un travail ne peut ici s'exécuter que sur un processeur à la fois, il n'y a pas d'exécution **parallèle** d'une même tâche possible.

Également :

- Stratégies hybrides : semi-partitionné, par exemple, seulement certaines tâches peuvent migrer et éventuellement uniquement sur sur deux processeurs au plus.
- Clusters : les processeurs sont partitionnés (regroupés) dans des clusters et les tâches sont assignées à des clusters. Leurs travaux peuvent par exemple migrer mais uniquement entre les processeurs d'un même cluster.

Classification Multipro. : Migrations/Préemptions

Si on prend les 3 classes d'assignation de priorité (et les 3-classes de migration traditionnelles), on obtient neuf configurations possibles.

Carpenter et al. [5] fournissent un tableau récapitulatif des relations entre-elles.

Relations importantes :

- L'ordonnancement partitionné n'est pas un cas particulier de l'ordonnancement global
- Il y a des systèmes ordonnançables avec des stratégies globales mais pour lesquels aucun partitionnement n'existe. Par exemple avec l'ensemble $\{\tau_1(1,2,2), \tau_2(2,3,3)$ et $\tau_3(2,3,3)\}$ sur 2 processeurs.
- En FTP, la réciproque est également vraie.

The winner is...

C'est la combinaison des **stratégies globales à migration totale** et **des priorités dynamiques** qui domine l'ensemble.

Classification des algorithmes d'ordonnancement multiprocesseur

Les comparaisons précédentes permettent de hiérarchiser les différentes classes d'ordonnancement multiprocesseur sur leur capacité à ordonnancer les ensembles.

Questions. Au sein d'une même classe :

- Pour les algorithmes optimaux, y-a-t'il d'autres critères d'intérêt (complexité, nb de migrations, de préemptions, etc.).
- Pour les algorithmes non-optimaux, comment les compare-t-on ?

Approches théoriques

- Bornes d'utilisation
- Comparaison indirecte vis-à-vis de l'optimal
  - Ratio asymptotique d'approximation
  - Facteur d'accélération

Évaluations empiriques

- Implantation sur un vrai système/application (patch LITMUS)
- Cas d'étude
- Générations de systèmes synthétiques et simulation

Augmentation de ressource

Définition (Facteur d'accélération)

Un algorithme d'ordonnancement a un facteur d'accélération (speedup factor) f, $f \geq 1$, s'il est capable d'ordonnancer n'importe quel ensemble de tâches ordonnançable par un algorithme optimal sur une plateforme donnée de processeurs identiques de vitesse 1, à condition que chaque processeur soit f fois plus rapide.

Théorème (Philipps et al., 1997)

Le facteur d'accélération de Global EDF est de $(2 - \frac{1}{m})$.

Par exemple, si un algorithme optimal est capable d'ordonnancer un ensemble de tâches sur 2 processeurs, vous êtes certains de pouvoir l'ordonnancer avec 2 processeurs 1.5 plus rapides sous Global EDF.

Évaluation empirique

Constat : Les outils de comparaison théoriques permettent de comparer les algorithmes mais sont généralement basés sur des pire-cas [6].

Ils ne donnent pas nécessairement d'information sur le comportement moyen de celui-ci.

Génération synthétique d'ensemble de tâches

- paramètres : distribution des périodes, échéances, des utilisations par tâche : aléatoire non-biaisé (UUnifast-Discard), benchmarks
- Qu'est-ce que l'on fait varier ? utilisation totale, nombre de tâches, etc.

Simulation

- Qu'est-ce l'on mesure : *Success ratio* (ordo. ou non ?), nombre de migrations, préemptions, temps de réponses, temps d'exécution de l'algo., etc.
- Combien de fois ?

Représentation : choisir une présentation des résultats adaptée (graphes, box-plot, histogrammes, etc.). Factoriser si possible : ***weighted schedulability***

Intérêt

- Varier un paramètre (nb de tâches par ex.) de concert avec l'utilisation
- Passer de 3D à 2D
- Éviter les biais (une utilisation basse favorise l'ordonnançabilité)

$$W_y(p) = \frac{\sum_{\forall \mathrm{T}} (U(\mathrm{T}) \cdot S_y(\mathrm{T}, p))}{\sum_{\forall \mathrm{T}} U(\mathrm{T})}$$

- $W_y(p)$: succès pondéré en fonction d'un paramètre p
- $S_y(\mathrm{T}, p)$: fonction valant 1 si T ordonnançable 0 sinon

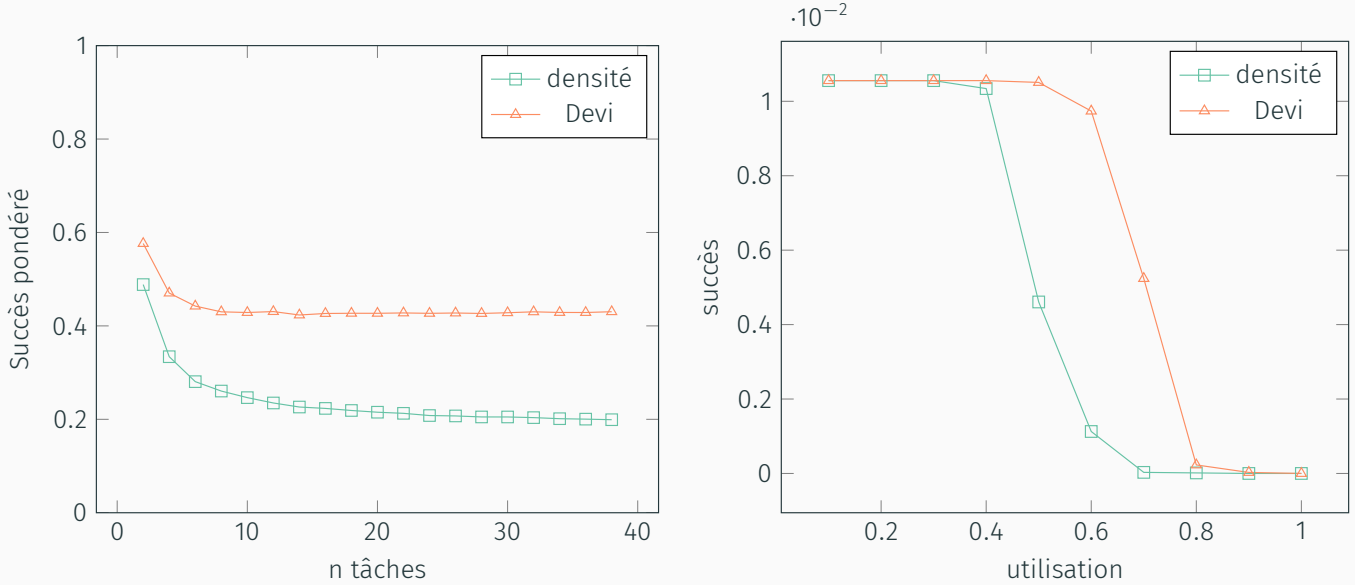


Fig. 5 : Comparaison du test de densité et du test de Devi (mono)

Évaluation empirique

Conseil

Veillez à bien décrire la méthode et veiller à la **reproductibilité**. Cela donne confiance dans vos résultats et permet de les répéter et les vérifier.

Tendance grandissante dans les conférences ⇒

Multipro. Partitionné

Approche Partitionné

1. Une étape **hors-ligne** de partitionnement des tâches sur les processeurs
2. L'application d'un algorithme d'ordonnancement monoprocesseur **en-ligne** sur chacun des processeurs.

- Le problème de partitionnement est un problème de **Bin-Packing** où chaque boîte représente un processeur dont la taille est déterminée par une condition d'ordonnançabilité (utilisation maximale de 1 pour EDF par ex.) et les objets des tâches dont la taille correspond à leur utilisation.
- Le Bin-Packing est un problème **NP-difficile**. On utilise donc des **heuristiques** pour approcher une solution optimale.

Partitionnement

1. **Tri** des tâches (objets)
   - Souvent par utilisation u_i décroissante des tâches
2. Règle de **placement** des tâches dans les processeurs (boîtes)
   - First-Fit (FF)
   - Next-Fit (NF)
   - Best-Fit (BF)
   - Worst-Fit (WF)

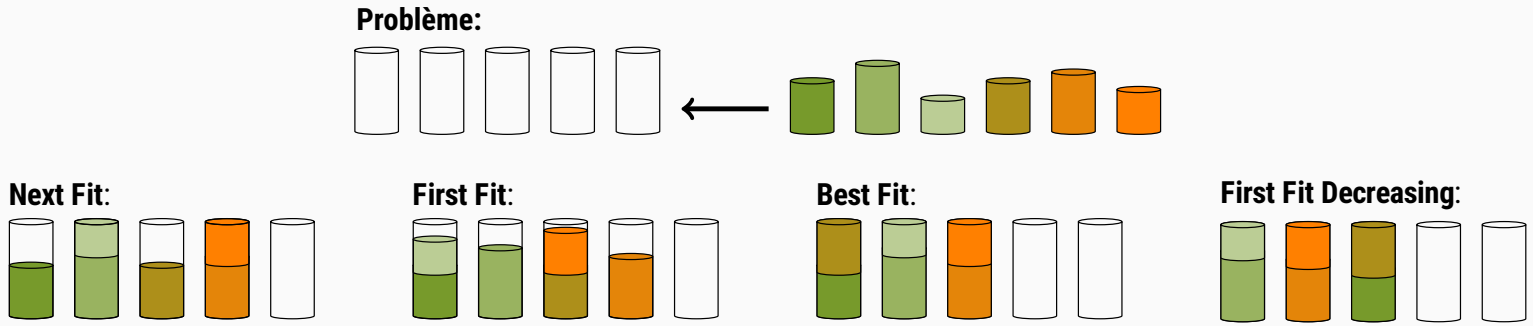


FIG. 6 : Des heuristiques de partitionnement

Ensuite, ordonnancement monprocesseur sur les m processeurs

Exemple : RMNF

Exemple de Rate Monotonic Next-Fit [7]

1. Les n tâches sont triées par ordre non-décroissant de leurs périodes
2. $i \leftarrow j \leftarrow 1$
3. Assignation
   - On assigne τ_i au processeur P_j **si** $\sum_{\tau_k \in P_j} \frac{C_k}{T_k} + C_i/T_i \leq 0.69$
   - **sinon**, on assigne τ_i au processeur P_{j+1}, $j \leftarrow j + 1$
4. Si $i < n$, on passe à la tâche suivante, $i \leftarrow i + 1$, puis on retourne à l'étape 3.
5. On a terminé, il faut j processeurs pour ordonnancer cet ensemble de tâches, on applique RM sur chacun d'eux.

Multiprocesseur - Partitionné : Ratio d'approximation

Il n'existe pas d'algorithme praticable donnant solution optimale.

- Un outil de **comparaison** des heuristiques : le ratio asymptotique d'approximation $\frac{\mathsf{A}(\mathrm{T})}{\mathsf{OPT}(\mathrm{T})}$ compare le nombre $\mathsf{A}(\mathrm{T})$ de processeurs requis pour ordonnancer n'importe quel ensemble de tâches T avec l'algorithme A et $\mathsf{OPT}(\mathrm{T})$ le nombre optimal (minimum) de processeurs requis pour ordonnancer T, lorsque $\mathsf{OPT}(\mathrm{T}) \to \infty$.

Par exemple, EDF-FF-(*) a un tel ratio de 1.7 et RM-FF de 2.67 .

Intérêt pratique limité : déterminer $\mathit{OPT}(\mathrm{T})$ est NP-difficile.

Multiprocesseur - Partitionné : Bornes d'utilisation

Théorème (Andersson et al [8])

Aucun algorithme partitionné/global FTP ou FJP ne peut garantir une borne d'utilisation supérieure à $U_{OPT} = (m+1)/2$

Pensez à $m+1$ tâches $\tau_i(1+\epsilon, 2)$, il est clair que quelle que soit la méthode de partitionnement, ou assignation en global, FTP/FJP, ça n'est pas faisable sur m processeurs.

Global non-optimal

Transposition du monoprocesseur

La transposition directe des politiques monoprocesseurs produit les politiques multiprocesseurs Global FTP (G-FTP) et Global EDF (G-EDF) en appliquant les règles suivantes :

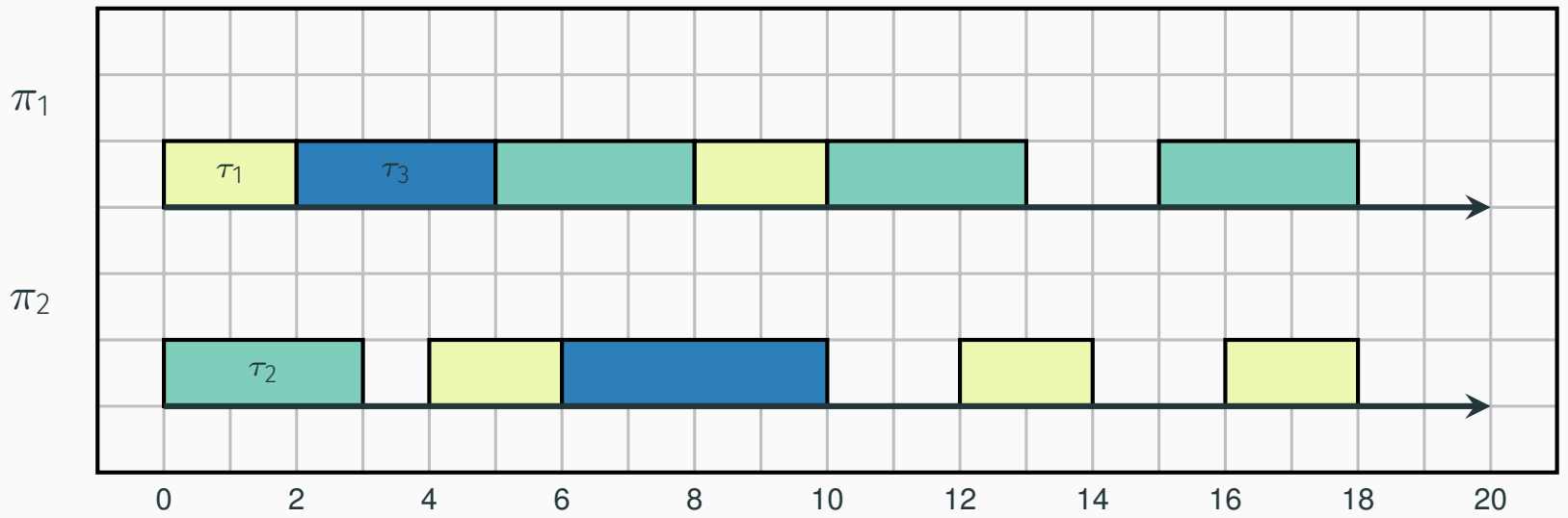
- Les priorités sont accordées selon FTP(RM, DM,etc.)/FJP (EDF)
- Les m travaux les plus prioritaires sont exécutés sur les m processeurs.

Sniff

RM/DM et EDF perdent leur optimalité adaptées en multiprocesseur.

G-RM : Exemple

Exemple d'ordonnancement selon Global-RM avec $\tau_1(2, 4, 4)$, $\tau_2(3, 5, 5)$, et $\tau_2(7, 16, 16)$ sur 2 processeurs.



Anomalies

Périodes : plusieurs politiques d'ordonnancement et tests ne sont pas T-viables (G-RM par ex., et en partitionné également) [8].
Instant critique : le pire temps de réponse d'une tâche ne correspond pas nécessairement au réveil simultané des tâches plus prioritaires.

Exemple : $\tau_1(2,2,8)$, $\tau_2(2,2,10)$, $\tau_3(4,6,8)$ et $\tau_4(4,7,8)$ selon G-DM.

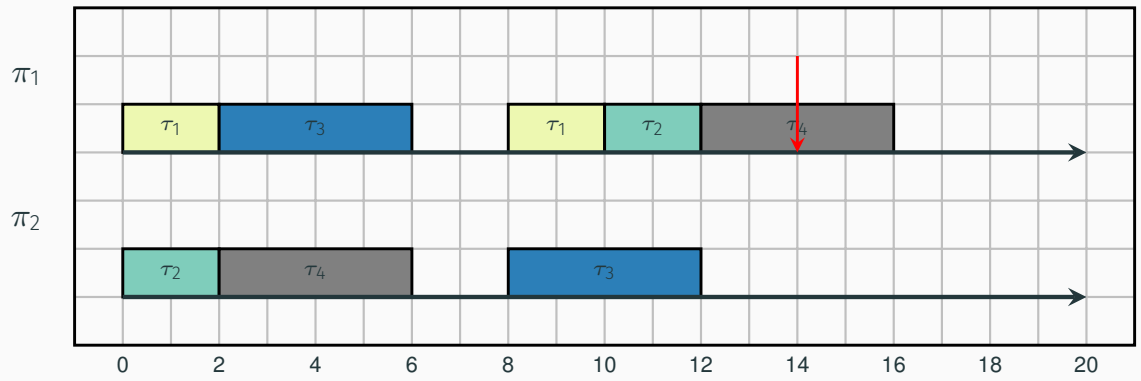


Fig. 7 : Tiré de [9]. τ_4 manque sa deuxième échéance mais pas la première.

Vérification

- Analyses **(tests suffisants FTP/FJP)** : interférence des *carry-(in,out) jobs*, voir [9, 10].
- Simulation : intervalle de faisabilité

Théorème (Cucu et Goossens 2006 [11])

$(0, H]$ *est un intervalle de faisabilité pour les ensembles de tâches* ***périodiques****, synchrones, à échéances contraintes ordonnancé par un algorithme multiprocesseur FTP ou FJP déterministe et sans mémoire.*

Exemple

- m processeurs, $m+1$ tâches à échéances implicites.
- $\tau_1, \ldots, \tau_m(2\epsilon, 1, 1)$ et $\tau_{m+1}(1, 1+\epsilon, 1+\epsilon)$

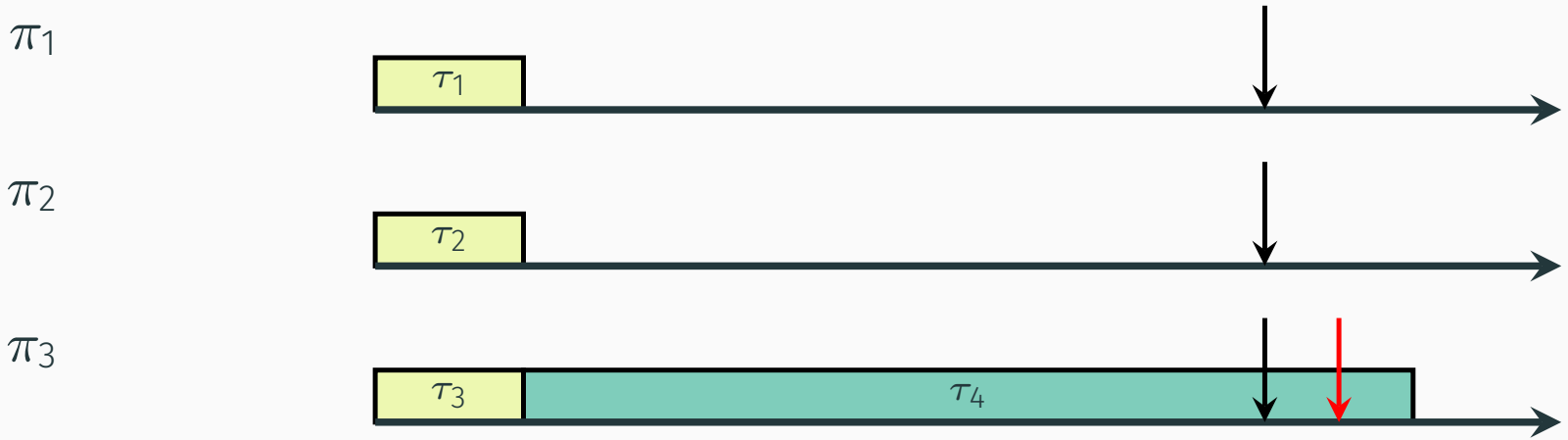


FIG. 8 : Pour $m = 3$, τ_4 manque son échéance à $t = 1 + \epsilon$

- G-RM/DM et G-EDF ne savent pas ordonnancer cet ensemble qui a pourtant un U très proche de 1 ($U \approx u_{max} = \max_{\forall i} \left(\frac{C_i}{T_i}\right)$).
- La tâche pénalisante est τ_{m+1}, elle est considérée comme lourde.

Sur cette observation, des algorithmes/tests d'ordonnancement ont tenté d'accorder un traitement particulier à ce type de tâches.

Par exemple pour G-EDF et tâches sporadiques à échéances implicites : borne **serrée** GFB [12] de $U_{GFB} = m - (m-1)u_{max}$ où u_{max} est l'utilisation maximale d'une tâche

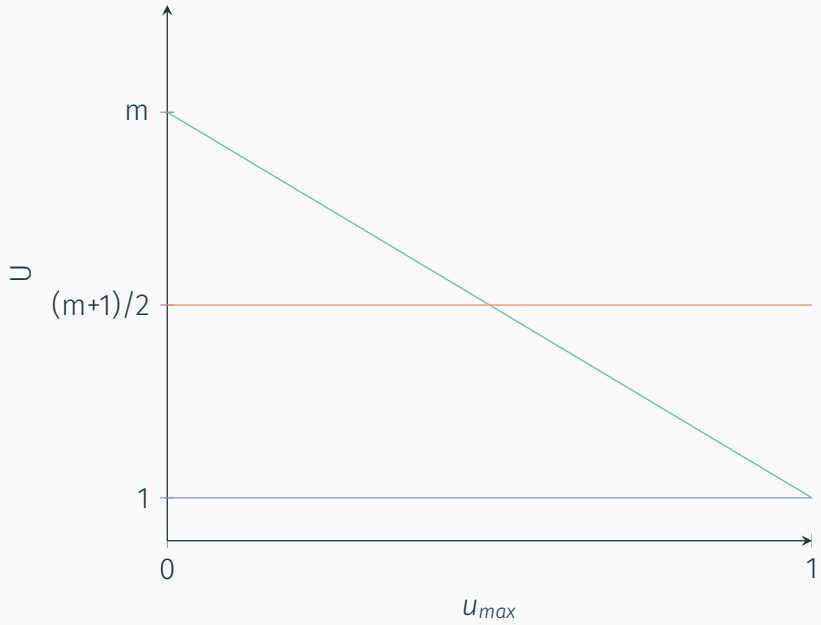


Fig. 9 : Borne GFB

Approches Utilization Threshold (US)

Principe

- On définit un seuil d'utilisation η puis
- Si $u_i > \eta$, on attribue une priorité maximale (infinie) à τ_i
- Sinon la priorité de τ_i est assignée selon l'algorithme original (RM/EDF)
- Les ex æquo sont réglés arbitrairement

Performances

- Toujours non optimaux
- Mais dominent les versions classiques

Quelques résultats (échéances implicites)

- Borne [13] $U_{EDF-US[1/2]} = (m+1)/2$ (maximum possible)
- Borne [8] $U_{RM-US[m/(3m-2)]} = m^2/(3m-2)$
- Borne maximale G-FTP[η] [14] $U \leq (\sqrt{2}-1)m \approx 0.41m$

Principe de EDF$^{(k)}$ [12]

1. Tri des tâches par utilisation non-croissante.
2. Assignation de la priorité maximale (échéance absolue $-\infty$) aux $k-1$ premières tâches.
3. Les autres tâches sont gérées classiquement selon G-EDF.

Un ensemble de tâches sporadiques à échéances implicites est ordonnançable sur m processeurs si

$$m \geq (k-1) + \left\lceil \frac{\sum_{i=k+1}^{n} u_i}{1-u_k} \right\rceil$$

Remarques

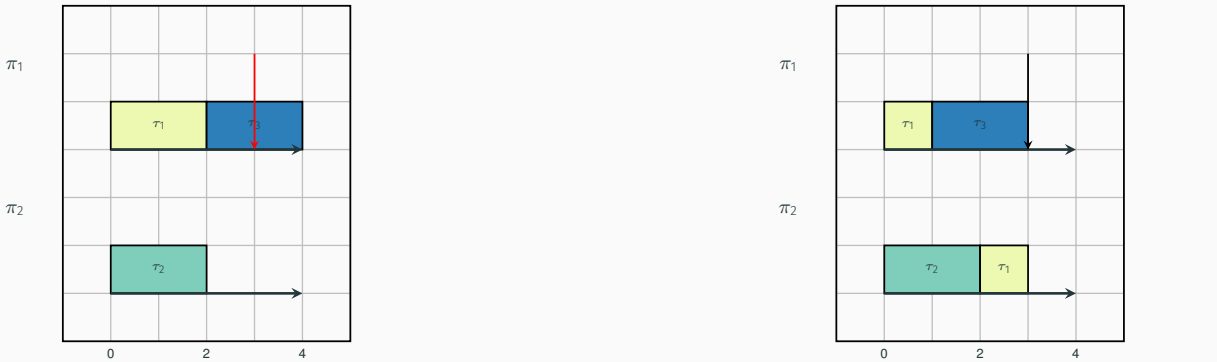
- EDF$^{(1)}$ correspond à G-EDF.
- EDF$^{(k_{min})}$ domine EDF-US[1/2], k_{min} étant le k qui minimise la partie droite de l'équation ci-dessus.

Un pas de plus vers l'optimalité ?

La version globale de LLF ne domine pas G-EDF mais **EDZL (ZL pour zero-laxity) oui !**

Principe de EDZL : G-EDF sauf si un travail atteint une laxité nulle et acquiert alors la priorité la plus élevée.

- Borne inférieure d'utilisation de $\frac{m+1}{2}$ (échéances implicites)
- Principe étendu au FTP avec FPZL



(a) G-EDF **(b)** EDZL

FIG. 10 : $\tau_1(2,3,3), \tau_2(2,3,3), \tau_3(2,3,3), m = 2$

EDZL

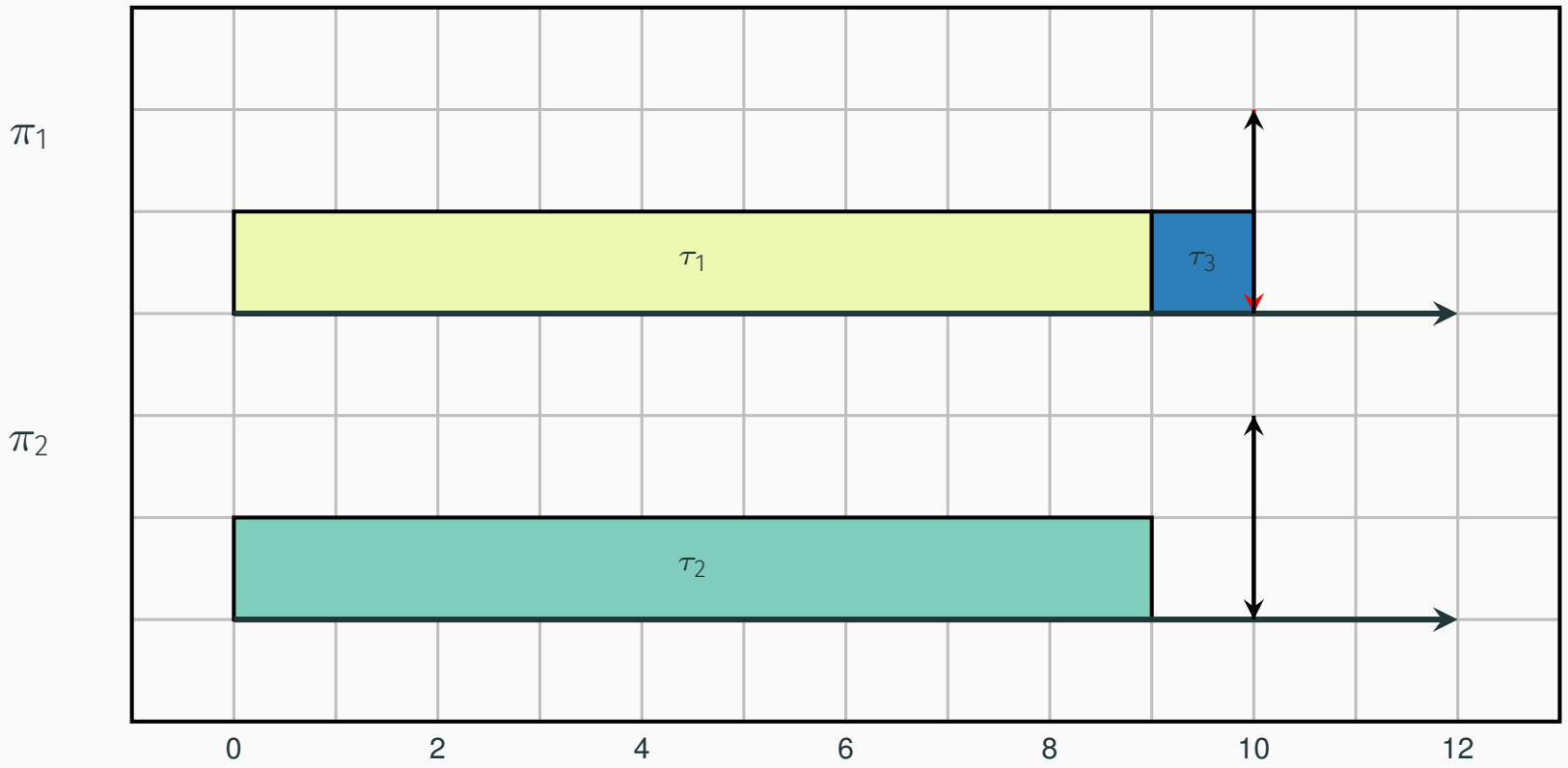


Fig. 11 : EDZL n'est pas optimal. Exemple avec $\tau_1(9, 10, 10), \tau_2(9, 10, 10), \tau_3(8, 40, 40), m = 2$

Global optimal

Théorème (Horn [15])

Tout ensemble de tâches sporadiques, synchrones, à échéances implicites est faisable sur une plateforme de m processeurs identiques de capacité 1 si et seulement si :

$U \leq m$ *et* $u_{max} \leq 1$.

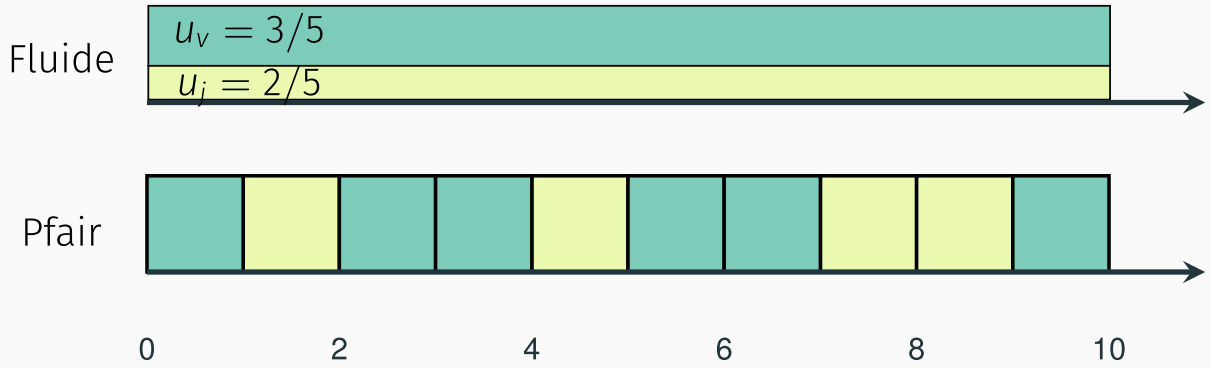
Sketch de preuve

Dans la preuve, ceci suppose de pouvoir préempter des tâches à des moments arbitraires et d'exécuter des tâches pendant des durées arbitrairement petites.

Les approches PFair sont des algorithmes optimaux dans le contexte du théorème ci-dessus et applicables dans le cas discret.

Global optimal : Approches Pfair

Ordonnancement idéal dit **fluide** vs. approche **basée quantum** (*quantum-based*).



- idéal (fluide) : Chaque tâche reçoit exactement $u_i \cdot t$ unités de temps processeur (donc proportionnellement à son utilisation) dans l'intervalle $[0, t)$. (Impossible dans le cas discret)
- basée qantum (Pfair) : Chaque tâche reçoit le nombre d'unité de temps (discret) processeur proportionnellement à son utilisation **au plus juste** de l'idéal.

Qu'est-ce que ça veut dire **au plus juste** ?

Notion de retard (*lag*) : Le retard est la différence entre le temps idéalement alloué (ideal) et le temps alloué par Pfair (alloc).

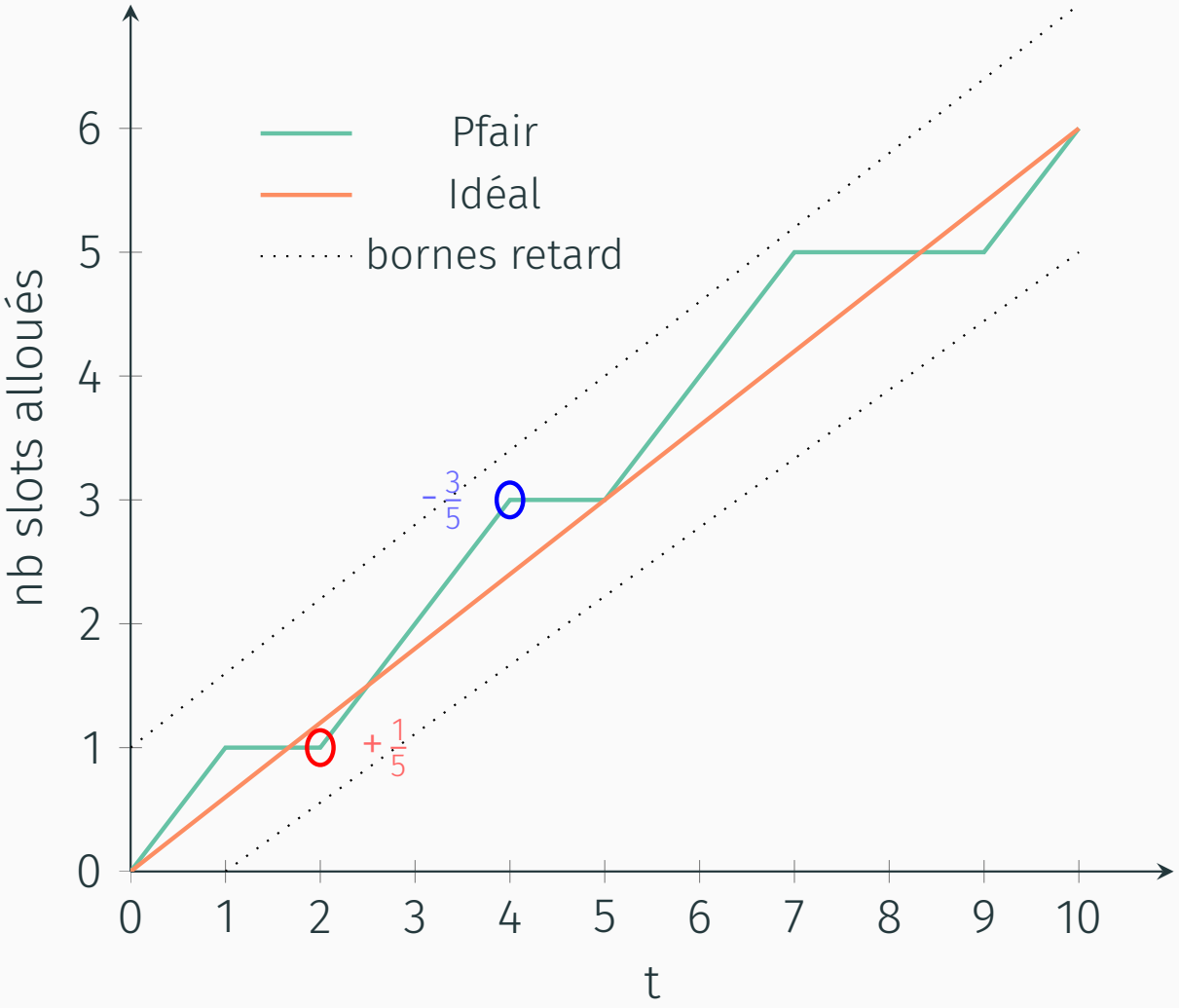
$$\begin{aligned} \text{retard}(\tau_i, t) &= \text{ideal}(\tau_i, t) - \text{alloc}(\tau_i, t) \\ &= u_i \times t - \text{alloc}(\tau_i, t) \end{aligned}$$

Définition (Ordonnancement Pfair [16] (1996))

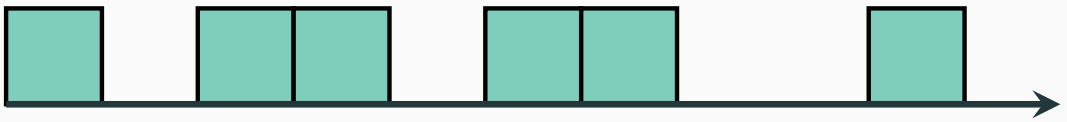
Un ordonnancement est Pfair si et seulement si
$-1 < \text{retard}(\tau_i, t) < 1, \forall \tau_i, t \in \mathcal{N}$.

Informellement, à chaque unité de temps, l'écart maximal entre le temps idéalement alloué et le temps alloué en pratique doit être inférieur à 1 unité de temps (quantum).

Global optimal : Approches Pfair



$u_V = \frac{3}{5}$



- $\textbf{retard}(\tau_V, 2) = 2 \cdot \frac{3}{5} - 1 = +\frac{1}{5} \Rightarrow$ sous-allocation
- $\textbf{retard}(\tau_V, 4) = 4 \cdot \frac{3}{5} - 3 = -\frac{3}{5} \Rightarrow$ sur-allocation

Global optimal : Approches Pfair

En pratique :

1. Chaque tâche est découpée en sous-tâches d'une unité de temps.
2. Pour chaque sous-tâche, on définit une date de réveil et une échéance (une fenêtre d'exécution) sous contrainte de retard.
3. Les priorités sont attribuées selon les échéances et l'exécution se fait suivant EDF (avec une règle pour les ex æquo).

Quelques algorithmes Pfair : PF, PD et PD^2.

Une variante : ER-Fair.

Global optimal : DP-Fair

Les approches PFair ont l'avantage du discret mais

- On doit prendre des décisions à chaque unité de temps.
- Il y a de fréquentes décisions d'ordonnancement, préemptions/migrations.
- Les processeurs doivent être synchronisés.

L'approche DP-Fair [17] est basée sur les constants suivants :

- il n'est pas nécessaire de "coller" autant à l'ordonnancement fluide.
- il suffit de ne pas avoir de retard à l'échéance de chaque tâche.

Global optimal : DP-Fair

DP-Fair correspond à Deadline-Partitioning-Fair

Définition (Deadline-Partitioning (DP))

Le DP consiste à découper le temps en fenêtres (slices) délimitées par les échéances absolues des travaux du système. A l'intérieur de chaque fenêtre, chaque travail se voit allouer une charge (workload) proportionnelle à son utilisation sur cette fenêtre, et toutes ces charges partagent la même échéance.

Soit $t_0 = 0, t_1, t_2, \ldots$ les échéances absolues distinctes des travaux des tâches du système avec $t_j < t_{j+1}, \forall j \geq 0$.

Principes DP-Fair

- Allocation : sur chaque fenêtre de longueur $L_j = t_j - t_{j-1}$, chaque tâche se voit allouer une charge proportionnelle à son utilisation $L_j \cdot u_i$. Par exemple, par l'application de la *McNaughton's wrap-around*.
- Ordonnancement sur chaque fenêtre résumé intuitivement comme il suite
  « *Si un travail doit s'exécuter pour terminer à temps (laxité nulle), fais-le. Si un travail a terminé, stoppe-le. N'autorise pas d'insertion de temps creux au delà de la marge (m − U) de l'ensemble de tâches.* »

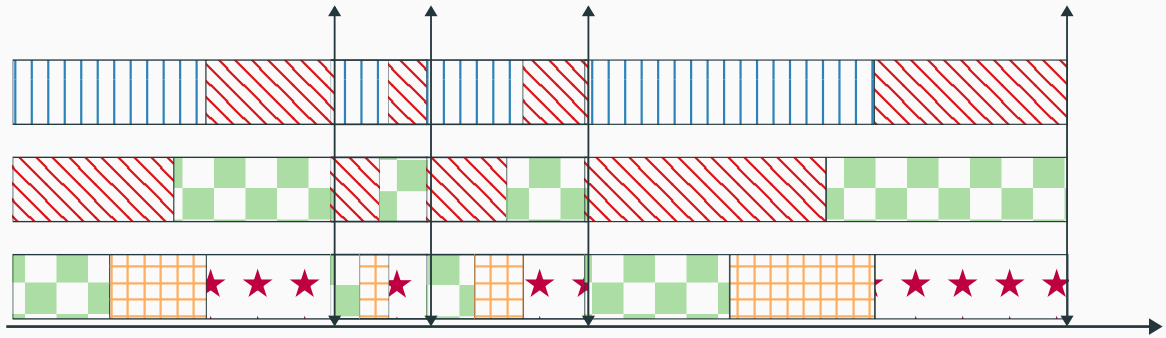


Fig. 12 : DP-Wrap est un exemple d'algorithme DP-Fair.

Global optimal : DP-Fair

Théorème (Optimalité des approches DP-Fair)

Tous les algorithmes DP-Fair sont optimaux pour des tâches périodiques à échéances implicites.

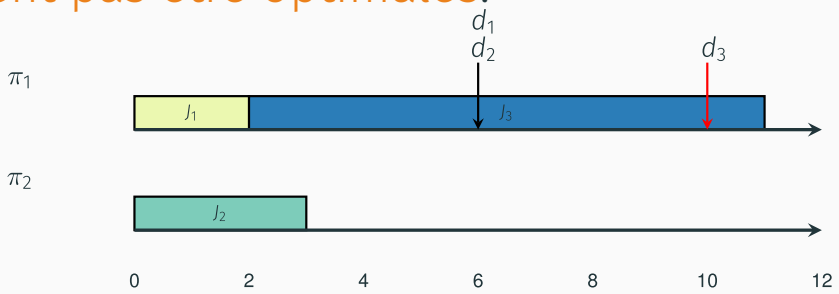
Avec des règles supplémentaires, les algorithmes DP-Fair sont optimaux pour des tâches sporadiques à échéances arbitraires.

Problème

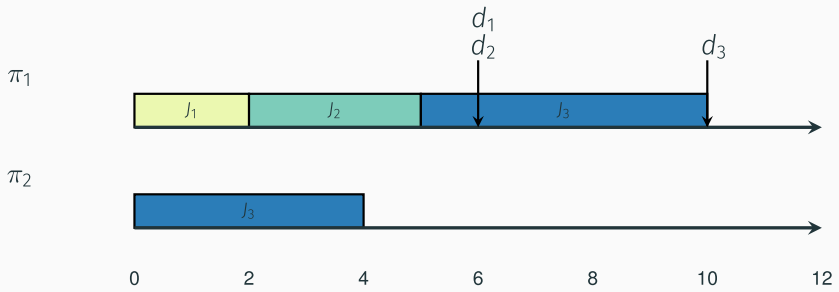
On est dans le domaine du continu, quid d'un fenêtre de taille 1 ?

Global optimal : U-EDF (2012)

L'algorithme U-EDF fait le même constat que DP-Fair : les approches gloutonnes ne peuvent pas être optimales.



(a) Approche verticale



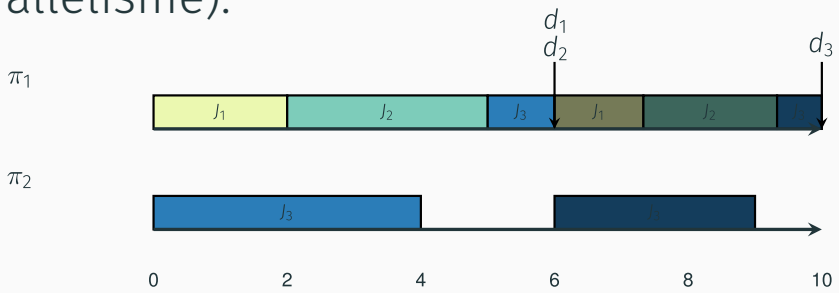
(b) Approche horizontale

FIG. 13 : Exemple tiré de l'article [18] avec 3 travaux $J_1(3,6), J_2(2,6), J_3(9,10)$ ($J_i(C_i, d_i)$, d_i est l'échéance absolue)

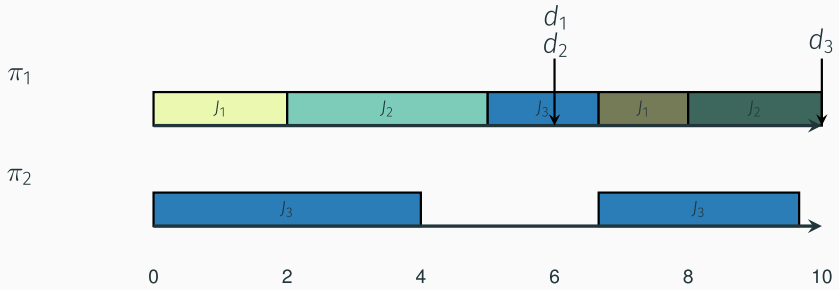
Global optimal : U-EDF (2012)

Intuitivement, **U-EDF**

- A chaque arrivée d'un travail : pré-allocation (réservation) sur les fenêtres suivantes (cf. DP-WRAP avec tri par échéances absolues croissantes) avec mise à jour du temps alloué à chaque tâche en fonction des exécutions déjà réalisées jusque-là.
- Ordonnancement EDF-D sur chaque processeur (variation de EDF pour empêcher le parallélisme).



(a) pré-allocations



(b) à $t = 6$, τ_1 et τ_2 ne sont pas arrivées $\Rightarrow$ re-calcul

Conclusion sur le global en processeurs identiques

Pour des processeurs identiques, tâches sporadiques à échéance implicite, les algorithmes optimaux capables d'atteindre une utilisation maximale de m ont les propriétés suivantes :

- Ordonnancement global sans restriction
- Priorité dynamique sur les travaux
- Présence de la *fairness* à un moment de l'histoire

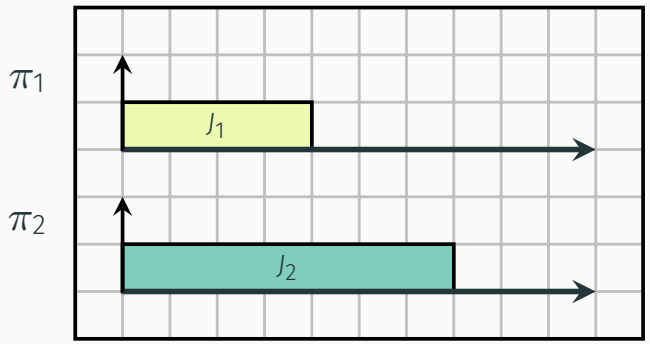
Techniques hybrides optimales modernes à étudier, à base de serveurs : RUN (périodique) et QPS [19] (sporadique).

Plateformes hétérogènes

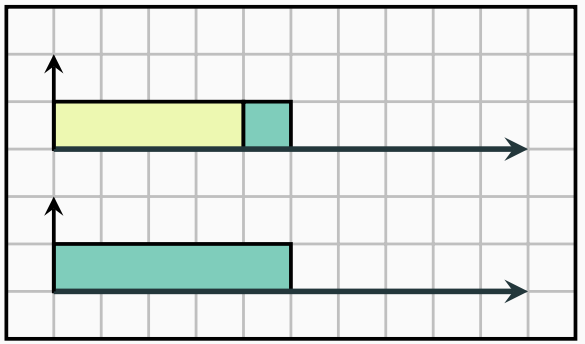
Plateformes uniformes : une vitesse par processeur

Beaucoup de résultats basés sur le **Level Algorithm** [20] algorithme optimal de l'ordonnancement "classique".

Intuition avec $C_1 = 8, C_2 = 7$ et $v_1 = 2, v_2 = 1$.



(a) Un travail par processeur



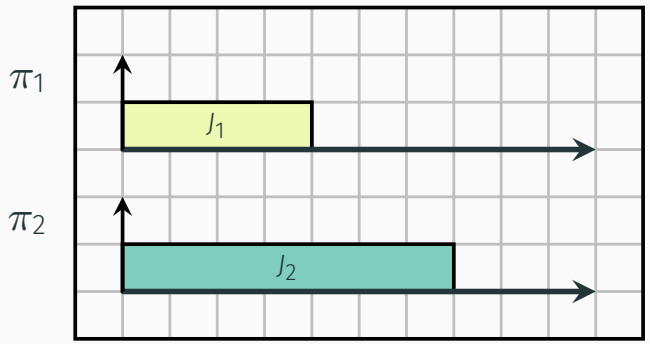
(b) Exécution à vitesse "moyenne" - makespan minimum

$$\text{makespan}_{min} = \max\left(\frac{C_1}{v_1}, \frac{C_1 + C_2}{v_1 + v_2}\right) = \frac{8+7}{2+1}$$

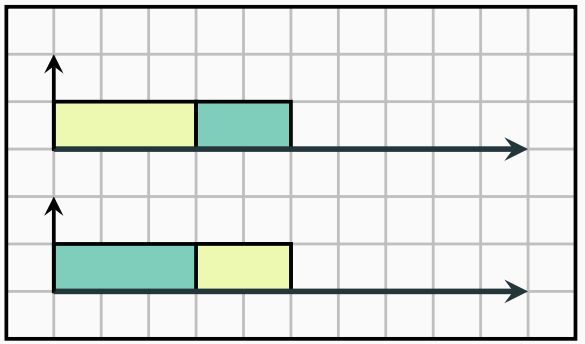
Plateformes uniformes : une vitesse par processeur

Beaucoup de résultats basés sur le **Level Algorithm** [20] algorithme optimal de l'ordonnancement "classique".

Intuition avec $C_1 = 8, C_2 = 7$ et $v_1 = 2, v_2 = 1$.



(a) Un travail par processeur

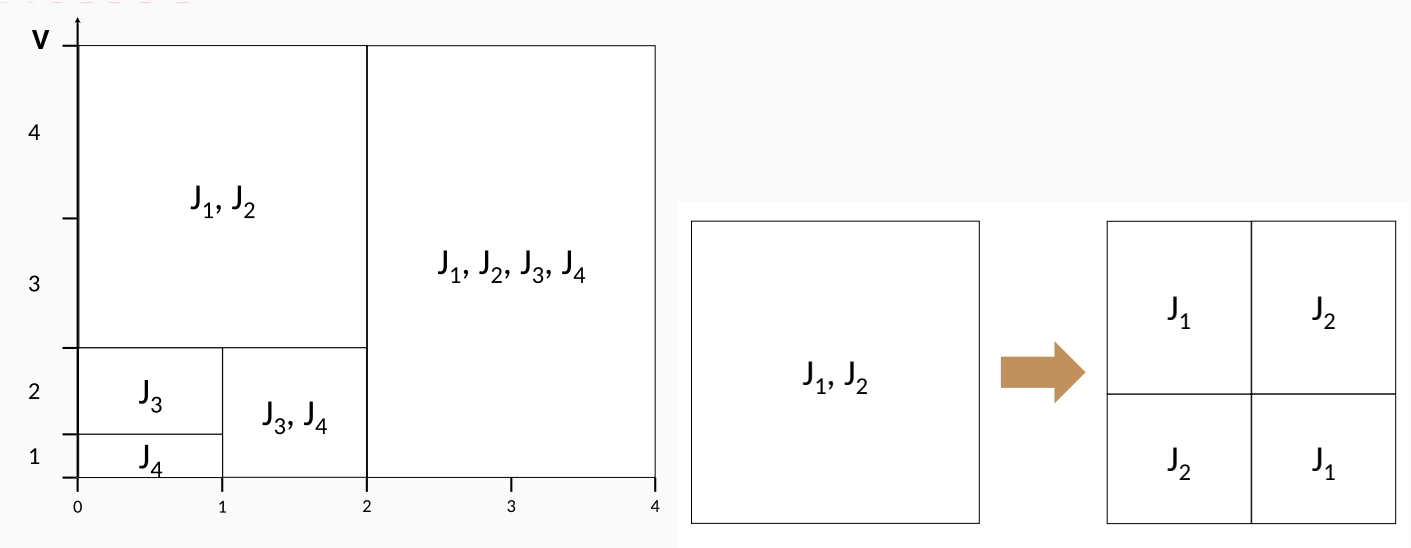


(b) Exécution à vitesse "moyenne" - makespan minimum - sans para.

$$\text{makespan}_{min} = \max\left(\frac{C_1}{v_1}, \frac{C_1 + C_2}{v_1 + v_2}\right) = \frac{8+7}{2+1}$$

Plateformes uniformes - Level Algorithm

- Toutes les tâches ayant le même *level* (temps d'exécution restant) sont exécutées conjointement
- Fonctionnement illustré avec 4 travaux
  $C_1 = 12, C_2 = 12, C_3 = 8.5, C_4 = 7.5$ sur 4 processeurs à vitesse variable
  $v_1 = 4, v_2 = 3, v_3 = 2, v_4 = 1$



(a) Ordonnancement abstrait (b) Ordonnancement concret

Fig. 18 : Level algorithm. Figures reprises de [21]

LLREF et LRE-TL [22], des politiques d'ordonnancement DP-Fair en uniforme s'en inspire (ainsi que de LLF).

Théorème (Funk et al. [23])

Un ensemble de tâches à échéances implicites est faisable sur une plateforme uniforme si et seulement si

$$U \leq S$$

$$AccU_k \leq S_k, k = 1, 2, \ldots, m.$$

avec $S = \sum_{v_j}, AccU_k = \sum_{i=1}^{k}, S_k = \sum_{j=1}^{k}$. Les processeurs et tâches sont triés par vitesse et temps d'exécution décroissants.

Plateformes unrelated

Unrelated : Une vitesse par couple tâche/processeur

- Modèle peu réaliste pour certaines applications
- Contexte des résultats : tâches périodiques, synchrones et à échéances implicites.
- Travaux fondateurs (faisabilité) Lawler et Labetoulle 1978 [24] (ordo. classique) puis Baruah 2004 [25] (ordo temps réel).
- Allocation linéaire et *mirroring* du *template schedule* à l'exécution, limité à 2 types (*Chwa et al. 2015*)

Un cas particulier : le **consistant** où il y a un ordre de grandeur entre les processeurs. Par exemple :

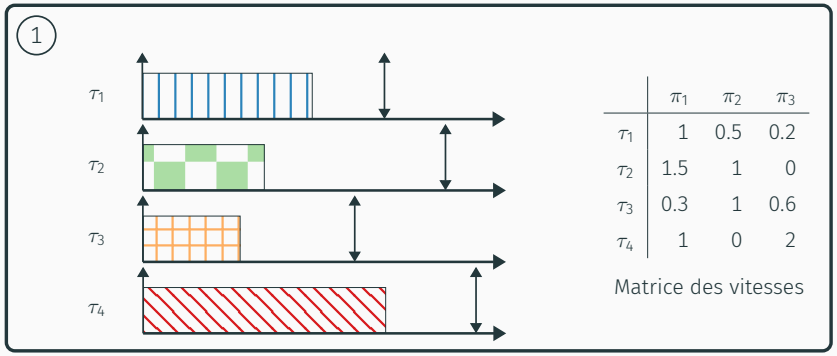
	π_1	π_2
τ_1	3	3/2
τ_2	4	1

Plateformes unrelated Principe

Stratégie (illustrée dans le transparent suivant)

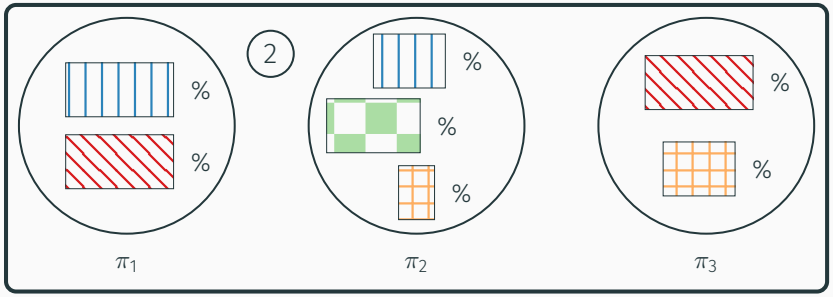
1. À partir de la matrice des vitesses et des tâches,
2. Calcul de l'allocation optimale des tâches sur les processeurs par résolution d'un LP (contraintes : capacité max des proc., tâches entièrement exécutées et pas de parallélisme),
3. Calcul d'un *template schedule* par des *matchings* successifs des tâches sur les processeurs en priorisant les tâches/processeurs importants,
4. Deadline partitionning et étirement du *template schedule* sur les fenêtres.

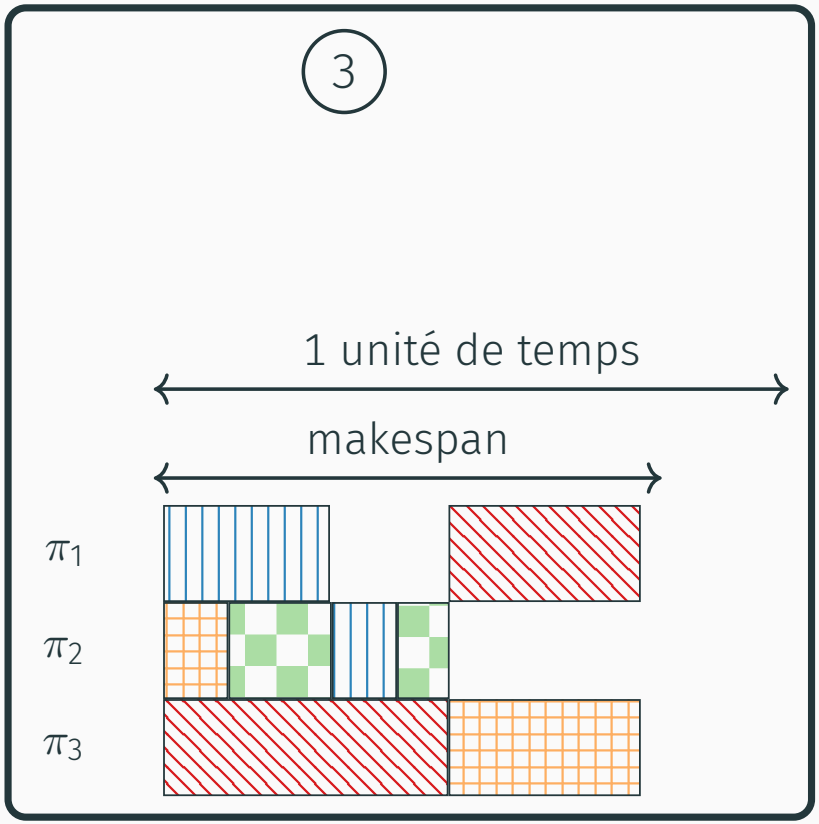
Plateformes unrelated Etapes

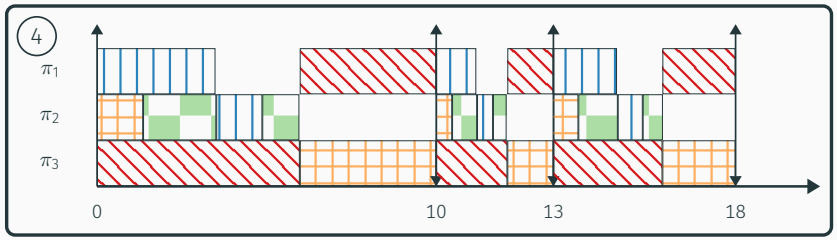


	π_1	π_2	π_3
τ_1	1	0.5	0.2
τ_2	1.5	1	0
τ_3	0.3	1	0.6
τ_4	1	0	2

Matrice des vitesses







Matrice X_c d'assignation de charge (workload assignment)

τ_i	$x_{i,1}$	$x_{i,2}$	$x_{i,3}$
τ_1	0.2	0.5	0.3
τ_2	0.7	0	0
τ_3	0	0.4	0.6
τ_4	0.1	0.1	0

$X_{1,2} = 0.5 \Rightarrow \pi_2$ exécute τ_1 50% de son temps.

Notion de présence

Le nombre de *présences* correspond au nombre de fois qu'une tâche est assignée sur un processeur.

Exemple : τ_1 a 3 présences (ou une présence de 3) tandis que τ_2 de 1
$\Rightarrow \tau_2$ est partitionnée.
$\Rightarrow \tau_1$ va nécessairement migrer.

Construction de la matrice bistochastique

A partir d'une matrix d'assignation de charge correcte X_c, il est possible de construire une matrice bistochastique B

$$B = \left(\begin{array}{c|c} X_c & B_n \\ \hline B_m & X_c^t \end{array}\right)$$

(B_n, B_m et X_c^t sont triviales à construire)

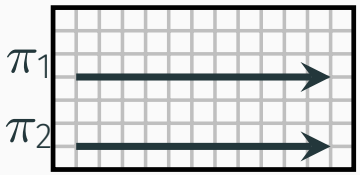
Définition (Matrice bistochastique)

Une matrice bistochastique ou doublement stochastique est une matrice carrée à coefficients réels positifs dont les sommes des éléments de chaque ligne et chaque colonne sont égales à 1.

Théorème (Birkhoff-von Neumann)

Une matrice bistochastique peut être décomposée en une combinaison linéaire de matrice de permutation.

Une décomposition BvN est similaire à une succession de **matchings** dans un graphe bipartite.

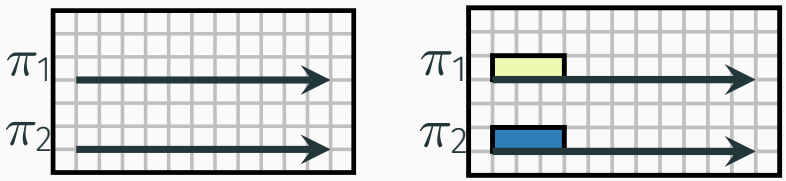


	π_1	π_2
τ_1	0.5	0.5
τ_2	0.4	0
τ_3	0	0.5

Théorème (Birkhoff-von Neumann)

Une matrice bistochastique peut être décomposée en une combinaison linéaire de matrice de permutation.

Une décomposition BvN est similaire à une succession de **matchings** dans un graphe bipartite.



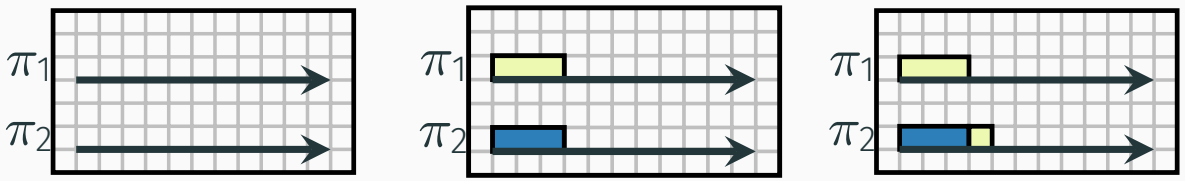
	π_1	π_2
τ_1	0.5	0.5
τ_2	0.4	0
τ_3	0	0.5

$$= \begin{pmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{pmatrix} \cdot 0.3 +$$

Théorème (Birkhoff-von Neumann)

Une matrice bistochastique peut être décomposée en une combinaison linéaire de matrice de permutation.

Une décomposition BvN est similaire à une succession de **matchings** dans un graphe bipartite.



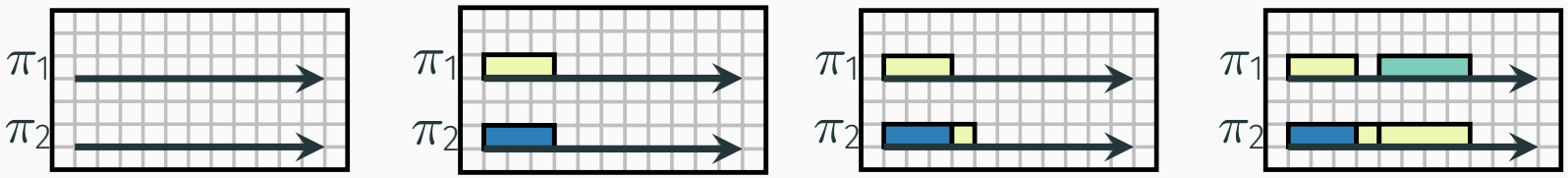
	π_1	π_2
τ_1	0.5	0.5
τ_2	0.4	0
τ_3	0	0.5

$$= \begin{pmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{pmatrix} \cdot 0.3 + \begin{pmatrix} 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{pmatrix} \cdot 0.1 +$$

Théorème (Birkhoff-von Neumann)

Une matrice bistochastique peut être décomposée en une combinaison linéaire de matrice de permutation.

Une décomposition BvN est similaire à une succession de **matchings** dans un graphe bipartite.



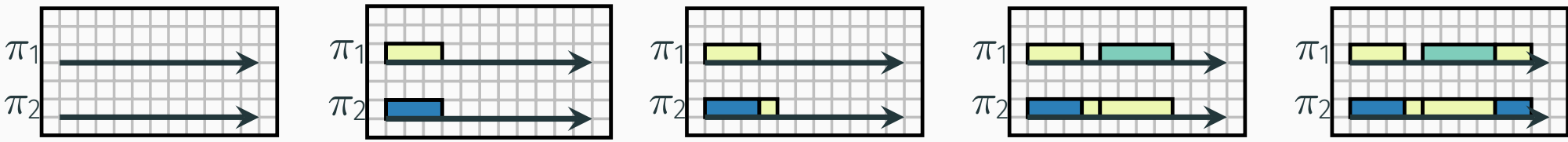
	π_1	π_2
τ_1	0.5	0.5
τ_2	0.4	0
τ_3	0	0.5

$$= \begin{pmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{pmatrix} \cdot 0.3 + \begin{pmatrix} 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{pmatrix} \cdot 0.1 + \begin{pmatrix} 0 & 1 \\ 1 & 0 \\ 0 & 0 \end{pmatrix} \cdot 0.4 +$$

Théorème (Birkhoff-von Neumann)

Une matrice bistochastique peut être décomposée en une combinaison linéaire de matrice de permutation.

Une décomposition BvN est similaire à une succession de **matchings** dans un graphe bipartite.



	π_1	π_2
τ_1	0.5	0.5
τ_2	0.4	0
τ_3	0	0.5

$$= \begin{pmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{pmatrix} \cdot 0.3 + \begin{pmatrix} 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{pmatrix} \cdot 0.1 + \begin{pmatrix} 0 & 1 \\ 1 & 0 \\ 0 & 0 \end{pmatrix} \cdot 0.4 + \begin{pmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{pmatrix} \cdot 0.2$$

Plan

Questions ?

Références

[1] Maxime Chéramy, Pierre-Emmanuel Hladik et Anne-Marie Déplanche. « SimSo : A Simulation Tool to Evaluate Real-Time Multiprocessor Scheduling Algorithms ». In : *Proc. of the 5th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems*. WATERS. 2014.

[2] Mathai Joseph et Paritosh Pandya. « Finding response times in a real-time system ». In : *The Computer Journal* 29.5 (1986), p. 390-395.

[3] Enrico Bini, Giorgio C Buttazzo et Giuseppe M Buttazzo. « Rate monotonic analysis : the hyperbolic bound ». In : *IEEE Transactions on Computers* 52.7 (2003), p. 933-942.

[4] Sanjoy Baruah, Marko Bertogna et Giorgio Buttazzo. *Multiprocessor scheduling for real-time systems*. Springer, 2015.

[5] John Carpenter et al. *A Categorization of Real-Time Multiprocessor Scheduling Problems and Algorithms*. 2004.

[6] Jian-Jia Chen et al. « On the pitfalls of resource augmentation factors and utilization bounds in real-time scheduling ». In : *29th Euromicro Conference on Real-Time Systems (ECRTS 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik. 2017.

[7] Yingfeng Oh et Sang H Son. *Preemptive scheduling of periodic tasks on multiprocessor : Dynamic algorithms and their performance*. Rapp. tech. Citeseer, 1993.

[8] Björn Andersson, Sanjoy Baruah et Jan Jonsson. « Static-priority scheduling on multiprocessors ». In : *Proceedings 22nd IEEE Real-Time Systems Symposium (RTSS 2001)(Cat. No. 01PR1420)*. IEEE. 2001, p. 193-202.

[9] Robert Davis et Alan Burns. « A survey of hard real-time scheduling algorithms and schedulability analysis techniques for multiprocessor systems ». In : *University of York, Department of Computer Science, techreport YCS-2009-443* (2009).

[10] Theodore P Baker. « Multiprocessor EDF and deadline monotonic schedulability analysis ». In : *RTSS 2003. 24th IEEE Real-Time Systems Symposium, 2003*. IEEE. 2003, p. 120-129.

[11] Liliana Cucu et Joël Goossens. « Feasibility intervals for fixed-priority real-time scheduling on uniform multiprocessors ». In : *2006 IEEE Conference on Emerging Technologies and Factory Automation*. IEEE. 2006, p. 397-404.

[12] Joël Goossens, Shelby Funk et Sanjoy Baruah. « Priority-driven scheduling of periodic task systems on multiprocessors ». In : *Real-time systems* 25.2 (2003), p. 187-205.

[13] Anand Srinivasan et Sanjoy Baruah. « Deadline-based scheduling of periodic task systems on multiprocessors ». In : *Information processing letters* 84.2 (2002), p. 93-98.

[14] Björn Andersson et Jan Jonsson. « The utilization bounds of partitioned and pfair static-priority scheduling on multiprocessors are 50% ». In : *15th Euromicro Conference on Real-Time Systems, 2003. Proceedings.* IEEE. 2003, p. 33-40.

[15] WA Horn. « Some simple scheduling algorithms ». In : *Naval Research Logistics Quarterly* 21.1 (1974), p. 177-185.

[16] Sanjoy K Baruah et al. « Proportionate progress : A notion of fairness in resource allocation ». In : *Algorithmica* 15.6 (1996), p. 600-625.

[17] Greg Levin et al. « DP-FAIR : A simple model for understanding optimal multiprocessor scheduling ». In : *2010 22nd Euromicro Conference on Real-Time Systems*. IEEE. 2010, p. 3-13.

[18] Geoffrey Nelissen et al. « U-EDF : An unfair but optimal multiprocessor scheduling algorithm for sporadic tasks ». In : *2012 24th Euromicro Conference on Real-Time Systems*. IEEE. 2012, p. 13-23.

[19] Ernesto Massa et al. « Quasi-partitioned scheduling : optimality and adaptation in multiprocessor real-time systems ». In : *Real-Time Systems* 52.5 (2016), p. 566-597.

[20] Edward C Horvath, Shui Lam et Ravi Sethi. « A level algorithm for preemptive scheduling ». In : *Journal of the ACM (JACM)* 24.1 (1977), p. 32-43.

[21] Kecheng Yang et James H Anderson. « An optimal semi-partitioned scheduler for uniform heterogeneous multiprocessors ». In : *2015 27th Euromicro Conference on Real-Time Systems*. IEEE. 2015, p. 199-210.

[22] Shelby Funk et Vijaykant Nanadur. « LRE-TL : An optimal multiprocessor scheduling algorithm for sporadic task sets ». In : *17th International Conference on Real-Time and Network Systems*. 2009, p. 159-168.

[23] Shelby Funk, Joël Goossens et Sanjoy Baruah. « On-line scheduling on uniform multiprocessors ». In : *Proceedings 22nd IEEE Real-Time*

Systems Symposium (RTSS 2001)(Cat. No. 01PR1420). IEEE. 2001, p. 183-192.

[24] Eugene L Lawler et Jacques Labetoulle. « On preemptive scheduling of unrelated parallel processors by linear programming ». In : *Journal of the ACM (JACM)* 25.4 (1978), p. 612-619.

[25] Sanjoy Baruah. « Feasibility analysis of preemptive real-time systems upon heterogeneous multiprocessor platforms ». In : *Real-Time Systems Symposium, 2004. Proceedings. 25th IEEE International*. IEEE. 2004, p. 37-46.