Ordonnancement temps réel multiprocesseur et réparti

Frank Singhoff

Bureau C-203

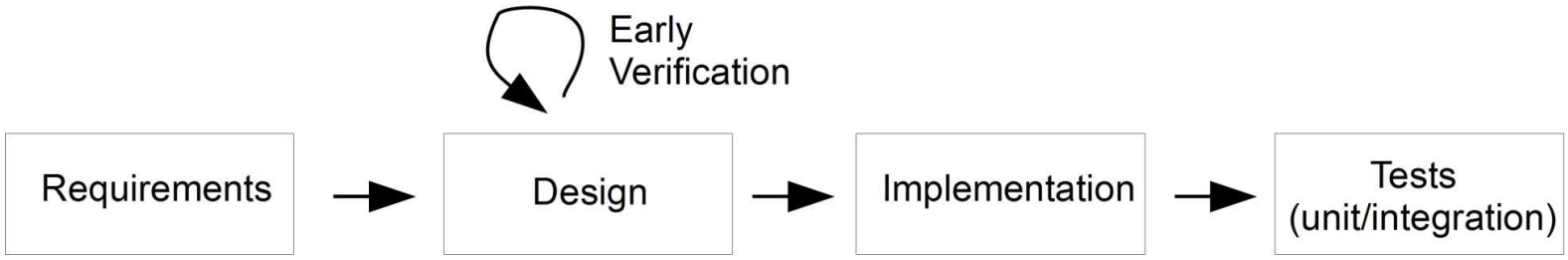
Université de Brest, France

Lab-STICC, CNRS UMR 6285

singhoff@univ-brest.fr

Sommaire

Vérifications amont d'un système



- **Etude NIST 2002:**
  - 70% des anomalies sont introduites lors de la phase de conception.
  - 3% d'entre elles trouvées lors de la conception. Coût : x1
  - Test unitaire : détection de 20% des anomalies. Coût : x5
  - Test d'intégration : détection de 10% des anomalies. Coût : x16

- **Objectifs:** trouver le plus d'anomalies lors de la phase de conception

=> Vérifications en amont.

Analyse d'ordonnancement (1)

- **Objectifs** : prédire le comportement temporel, prendre en compte les besoins d'urgence, d'importance des applications temps réel.

- **Taxinomie algorithmes d'ordonnancement :** préemptif/non préemptif, priorités statiques/dynamiques, hors/en ligne.

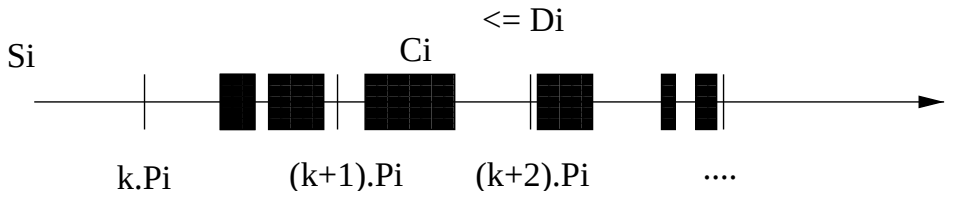
- **Principales propriétés recherchées :**

  1. **Facilité de mise en œuvre :** l'ordonnanceur est-il facile à implanter ?
  2. **Faisabilité :** est il possible d'exhiber un test de faisabilité ? Condition permettant de décider hors ligne du respect des contraintes des tâches.
  3. **Efficacité :** critères de comparaison des algorithmes $\Longrightarrow$ optimalité, dominance, équivalence et comparabilité.

Analyse d'ordonnancement (2)

- **Critères de comparaison des algorithmes:**
  - **Optimalité :** un algorithme a est optimal s'il est capable de trouver un ordonnancement pour tout ensemble faisable de tâches.
  - **Dominance** : a domine b si tous les jeux de tâche faisables par b le sont aussi par a et s'il existe des jeux de tâches faisables par a qui ne le sont pas par b.
  - **Équivalence** : a et b sont équivalents si tous les jeux de tâche faisables par a le sont aussi par b, et inversement.
  - **Incomparable** : a et b sont incomparables s'il existe des jeux de tâches faisables par a qui soient infaisables par b et inversement.

Analyse d'ordonnancement (3)



- **Paramètres définissant une tâche périodique i, tâche/fonction critique :**
  - Arrivée de la tâche dans le système : S_i.
  - Borne sur le temps d'exécution d'une activation, ou capacité, ou WCET : C_i.
  - Période d'activation : P_i. Notion de travail.
  - Délai critique : D_i (relatif à P_i).
- **Modèle tâches périodiques synchrones à échéances sur requêtes [LIU 73]** :
  - Tâches périodiques et indépendantes.
  - avec $\forall i : S_i = 0$ (instant critique, pire cas).
  - $\forall i : P_i = D_i$.

Exemple : algo. à priorité fixe (1)

- Priorités fixes $\Longrightarrow$ analyse hors ligne $\Longrightarrow$ applications statiques et critiques.

- **Propriétés/hypothèses :**
  - Tâches périodiques synchrones à échéances sur requêtes.
  - Complexité faible et mise en œuvre aisée.
  - Affectation Rate Monotonic : algorithme optimal dans la classe des algorithmes à priorité fixe. Dominé par EDF.

- **Fonctionnement :**

1. Affectation des priorités selon l'urgence ou l'importance. Ex : période (Rate Monotonic), délai critique (Deadline Monotonic).
2. Phase d'élection : élection de la plus forte priorité.

Exemple : algo. à priorité fixe (2)

1. **Intervalle de faisabilité** = $[0, PPCM(P_i)]$. Condition suffisante et nécessaire.
2. **Test sur le taux d'utilisation** (Rate Monotonic préemptif) :

$$U = \sum_{i=1}^{n} \frac{C_i}{P_i} \leq n(2^{\frac{1}{n}} - 1) \leq 0.69$$

   Condition suffisante mais non nécessaire.

3. **Test sur le pire temps de réponse** (qq soit l'affectation, préemptif) :

$$r_i = C_i + \sum_{j \in hp(i)} I_j = C_i + \sum_{j \in hp(i)} \left\lceil \frac{r_i}{P_j} \right\rceil C_j \leq D_i$$

   Condition généralement suffisante et nécessaire. $hp(i)$ est l'ensemble des tâches de plus forte priorité que i. Instant critique requis.

Exemple : algo. à priorité fixe (3)

- **Technique de calcul** : on évalue de façon itérative w_i^n par :

$$w_i^{n+1} = C_i + \sum_{j \in hp(i)} \left\lceil \frac{w_i^n}{P_j} \right\rceil C_j$$

- On démarre avec $w_i^0 = C_i$.
- Conditions d'arrêt :
  - Echec si $w_i^n > P_i$.
  - Réussite si $w_i^{n+1} = w_i^n$.

Exemple : algo. à priorité fixe (4)

- **Exemple :** P1=7 ; C1=3; P2=12 ; C2=2 ; P3=20 ; C3=5

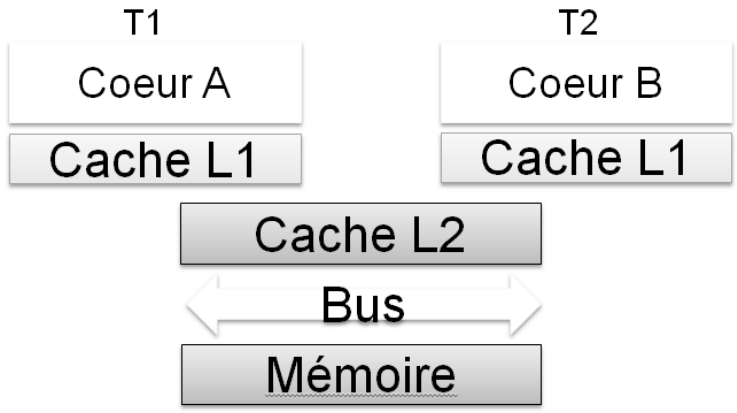
- $w_1^0 = 3 \Longrightarrow r_1 = 3$
- $w_2^0 = 2$
- $w_2^1 = 2 + \lceil \frac{2}{7} \rceil 3 = 5$
- $w_2^2 = 2 + \lceil \frac{5}{7} \rceil 3 = 5 \Longrightarrow r_2 = 5$
- $w_3^0 = 5$
- $w_3^1 = 5 + \lceil \frac{5}{7} \rceil 3 + \lceil \frac{5}{12} \rceil 2 = 10$
- $w_3^2 = 5 + \lceil \frac{10}{7} \rceil 3 + \lceil \frac{10}{12} \rceil 2 = 13$
- $w_3^3 = 5 + \lceil \frac{13}{7} \rceil 3 + \lceil \frac{13}{12} \rceil 2 = 15$
- $w_3^4 = 5 + \lceil \frac{15}{7} \rceil 3 + \lceil \frac{15}{12} \rceil 2 = 18$
- $w_3^5 = 5 + \lceil \frac{18}{7} \rceil 3 + \lceil \frac{18}{12} \rceil 2 = 18 \Longrightarrow r_3 = 18$

Architectures ciblées (1)

- **Historiquement :** plate-forme = un processeur **dédié**, une horloge et une mémoire commune ... prédictible mais cher et puissance bornée technologiquement.
- **Vers l'utilisation d'architectures plus puissantes, mais moins prédictibles :**
  - Utilisation de matériels sur étagère, ou processeur COTS, « Commercial Off The Shelf » (Federal aviation administration, 2011).
  - Plusieurs unités de calcul/processeurs identiques (ou non).
  - Partageant diverses ressources : cache, mémoire, bus, crossbar, NoC (Network-on-Chip) => **Nouvelles interferences**

Architectures ciblées (2)

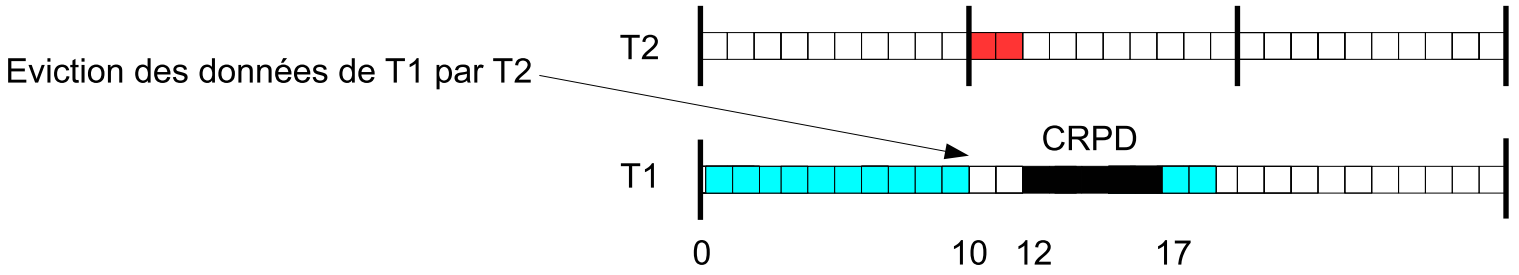
- **Architecture multicoeurs :**



- T1 et T2 sont fonctionnellement indépendantes ... mais finalement dépendantes à cause des ressources matérielles partagées : interférences !
- Une tâche peut être retardée à cause des contentions/interférences sur les ressources matérielles partagées (ex: cache).

Architectures ciblées (3)

- **Architecture multicoeurs :**

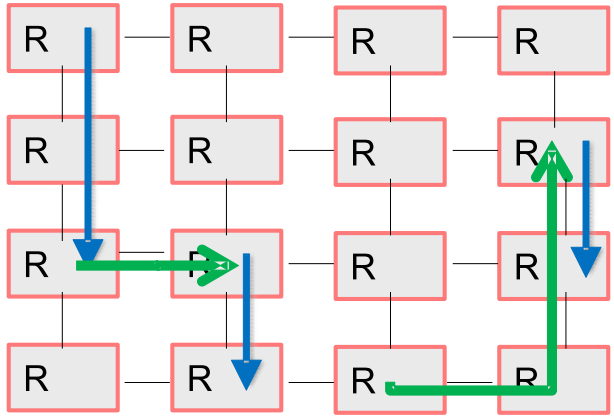


- **Cache related preemption delay (CRPD):** temps additionnel pour recharger le cache avec les blocs évincés lors d'une préemption => **interférence**.
- CRPD, peut représenter jusqu'à 44% du WCET (Pellizzoni 2007). Variabilité importante = analyse d'ordonnancement difficile.

Architectures ciblées (4)

- **Exemples de problème :**
  - Comment affecter les priorités avec des caches ? Rate Monotonic n'est plus optimal ici.
  - Comment simuler l'ordonnancement avec CRPD ? Dans le cas général avec CRPD:
    - La simulation d'ordonnancement est "non-sustainable".
    - L'intervalle de faisabilité n'est pas connu.
    - Sustainability : analyse pire cas valide qui reste valide dans les cas plus favorables.

Architectures ciblées (5)



- **Network-on-Chip :**
  - Routeurs, liens physiques, unités de calcul ou mémoire.
  - Interférence directe : flux qui utilisent le même lien physique au même moment.
  - Interférence indirecte : deux flux qui ne partagent aucun lien mais interfèrent via un troisième flux.

Architectures ciblées (6)

- "*Un système réparti est un ensemble de machines autonomes connectées par un réseau, et équipées d'un logiciel dédié à la coordination des activités du système ainsi qu'au partage de ses ressources." Coulouris et al. [COU 94].*

- **Pourquoi un système réparti ?**

1. Tolérance aux pannes (fiabilité, disponibilité).
2. Contraintes physiques (ex : avionique, usines automatisées).
3. Contraintes liées au processus industriel.
4. Optimisation partage des ressources.
5. ...

Architectures ciblées (7)

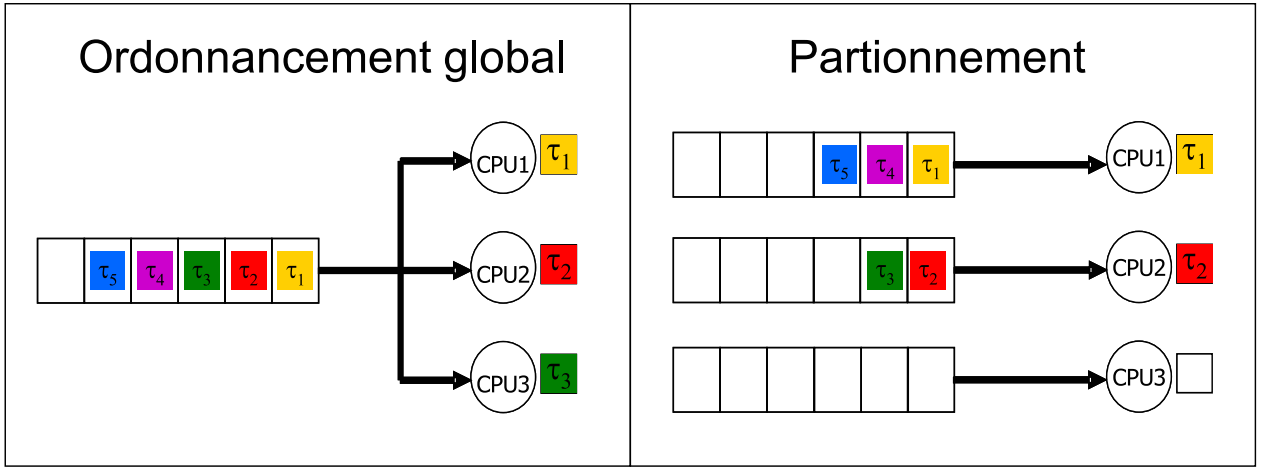
- **Exemples de problèmes soulevés par les systèmes répartis :**

1. Placement des tâches ? Paramètres des tâches (priorités) ?
2. Calcul temps acheminement des messages (accès au réseau) ?
3. Calcul délais de bout en bout émetteur/récepteur ?
4. ...

Architectures ciblées (8)

$\Longrightarrow$ Nécessite de nouvelles méthodes d'analyse de l'ordonnancement.

Approches d'ordonnancement (1)



- Ordonnancement dans un système réparti/multi-processeurs :

1. **Ordonnancement global :** choisir d'abord la tâche, puis placer la tâche sur un des processeurs libres. Approche en-ligne.
2. **Ordonnancement par partitionnement :** placement des tâches pour ordonnancement local, puis vérification éventuelle des délais de bout en bout. Approche hors-ligne.

Approches d'ordonnancement (2)

- **Ordonnancement global :**
  - Peu adapté aux systèmes hétérogènes et réparti : processus industriels/ingénierie de systèmes, migration de tâches/travaux.
  - Résultats théorie ordonnancement temps réel global récents, pour des architectures simples. Quid des ressources partagées ?
  - Meilleure optimisation des ressources : processeur occupé, moins de préemption ... mais des migrations.

$\Longrightarrow$ architectures multi-coeurs.

Approches d'ordonnancement (3)

- **Ordonnancement par partitionnement :**
  - Meilleure adéquation aux systèmes hétérogènes, de grande taille.
  - Théorie ordonnancement temps réel mono-processeur mature.
  - Utilisation moins optimale des ressources (processeur libre).
  - Isolation : retards/pannes/anomalies d'une tâche limités à un processeur.
  - Vérification de contrainte de bout en bout délicate. Comment intégrer les communications dans l'analyse.
  - Problème de Placement (Bin-packing, NP-difficile).

$\Longrightarrow$ architectures réparties hétérogènes.

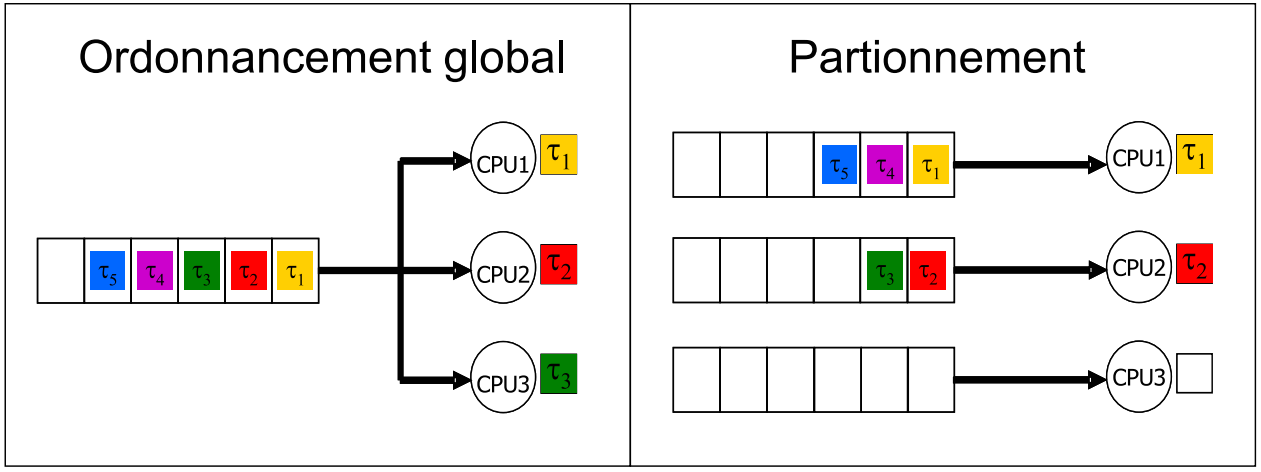
Approches d'ordonnancement (4)

- **Approches similaires pour (toute les ?) resources:**
  - Partitionnement de la ressource : prédictibilité ... mais sous utilisation des ressources.
  - Partage de la ressource : prédictibilité plus difficile ... mais ressources plus efficacement utilisée.
- **Autre exemple:** cache partitionné versus cache partagé
  - Partition trop petite : temps supplémentaire pour recharger les données
  - Partition trop grande : gaspillage de la ressource

Sommaire

1. Introduction et rappels.
2. Ordonnancement global, illustration avec les architectures multi-coeurs.
3. Ordonnancement par partitionnement, illustration avec les architectures réparties.
4. Ordonnancement par approches mixtes.
5. Résumé.
6. Acronymes.
7. Références.
8. Remerciements.

Approches d'ordonnancement



- Ordonnancement dans un système réparti/multi-processeurs :

1. **Ordonnancement global :** choisir d'abord la tâche, puis placer la tâche sur un des processeurs libres. Approche en-ligne.
2. **Ordonnancement par partitionnement :** placement des tâches pour ordonnancement local, puis vérification éventuelle des délais de bout en bout. Approche hors-ligne.

Ordonnancement global (1)

- **Types de plate-forme multiprocesseur :**
  - **Processeurs identiques :** processeur ayant la même capacité de calcul.
  - **Processeurs uniformes :** cadence d'exécution associée à chaque processeur tel, exécuter un travail de durée t sur un processeur de cadence c requiert $t \cdot c$ unités de temps.
  - **Processeurs spécialisés :** cadence d'exécution définie par le triplet $(r_{i,j}, i, j)$: le travail i requiert $r_{i,j} \cdot t$ unités de temps pour une durée de travail t sur le processeur j.

$\Longrightarrow$ On suppose ici, des processeurs identiques, et pas de ressource partagée (cache, bus mémoire).

Ordonnancement global (2)

- **Un ordonnanceur global traite deux problèmes [DAV 09, BER 07] :**

  - Quand et comment affecter les priorités des tâches/travaux.
  - Choisir le processeur où exécuter la tâche.

Ordonnancement global (3)

- **Instants de migration :**
  - Migration impossible : la tâche ne peut pas migrer. Tous ses travaux sont affectés à un processeur unique => partitionnement.
  - Migration possible entre les différents travaux d'une tâche. Un travail ne peut pas migrer.
  - Migration d'un travail possible d'une unité de temps à une autre (parallélisation d'un travail généralement interdit).
- **Affectations de priorité :**
  - Priorité fixe associée à la tâche (ex : RM).
  - Priorité fixe associée au travail (ex : EDF).
  - Priorité dynamique chaque unité de temps du travail/tâche (ex : LLF).

Ordonnancement global (4)

- **Deux types d'algorithmes**

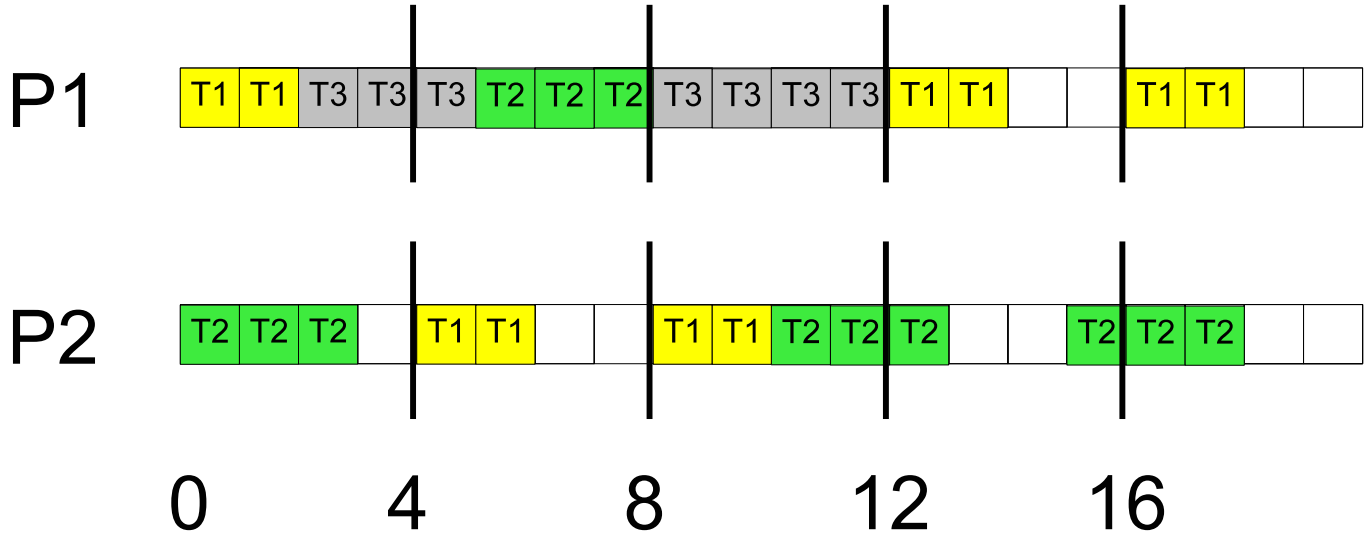
1. Adaptations d'algorithmes centralisés :
   - global RM, global EDF, global DM, global LLF, ...
   - Choix du niveau de migration
   - Appliquer globalement sur l'ensemble des processeurs une stratégie d'ordonnancement. Attribuer à chaque instant les m processeurs aux m tâches/travaux les plus prioritaires.
   - Préemption d'une tâche/travail lorsque tous les processeurs sont occupés.
2. Algorithmes nouveaux : PFair, EDZL, RUN, LLREF, EDF(k), SA,...

- Attention : domaine différent de l'ordonnancement mono-processeur => résultats/propriétés différents, résultats généraux moins nombreux.

Ordonnancement global (5)

- **Exemple** : global Deadline Monotonic

	Ci	Pi	Di
T1	2	4	4
T2	3	5	5
T3	7	20	20

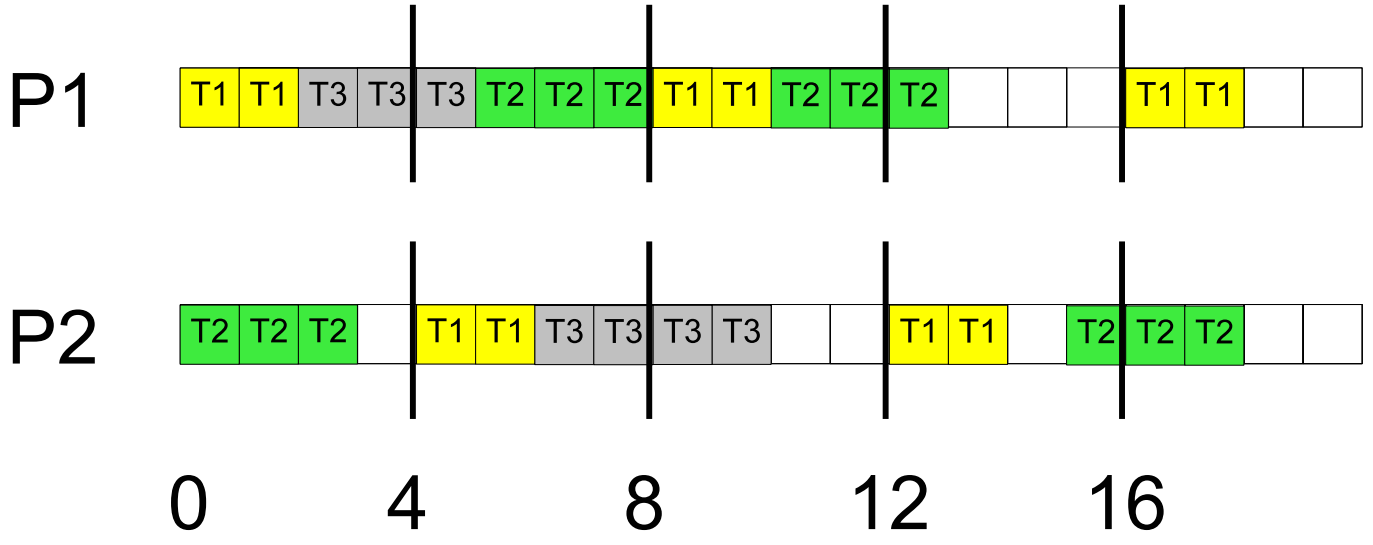


- Affectation des priorités : $T_1 > T_2 > T_3$.
- Migrations possibles des travaux.

Ordonnancement global (6)

- **Exemple** : global Deadline Monotonic

	Ci	Pi	Di
T1	2	4	4
T2	3	5	5
T3	7	20	20



- Affectation des priorités : $T_1 > T_2 > T_3$.
- Migrations possibles chaque unité de temps.

Ordonnancement global (7)

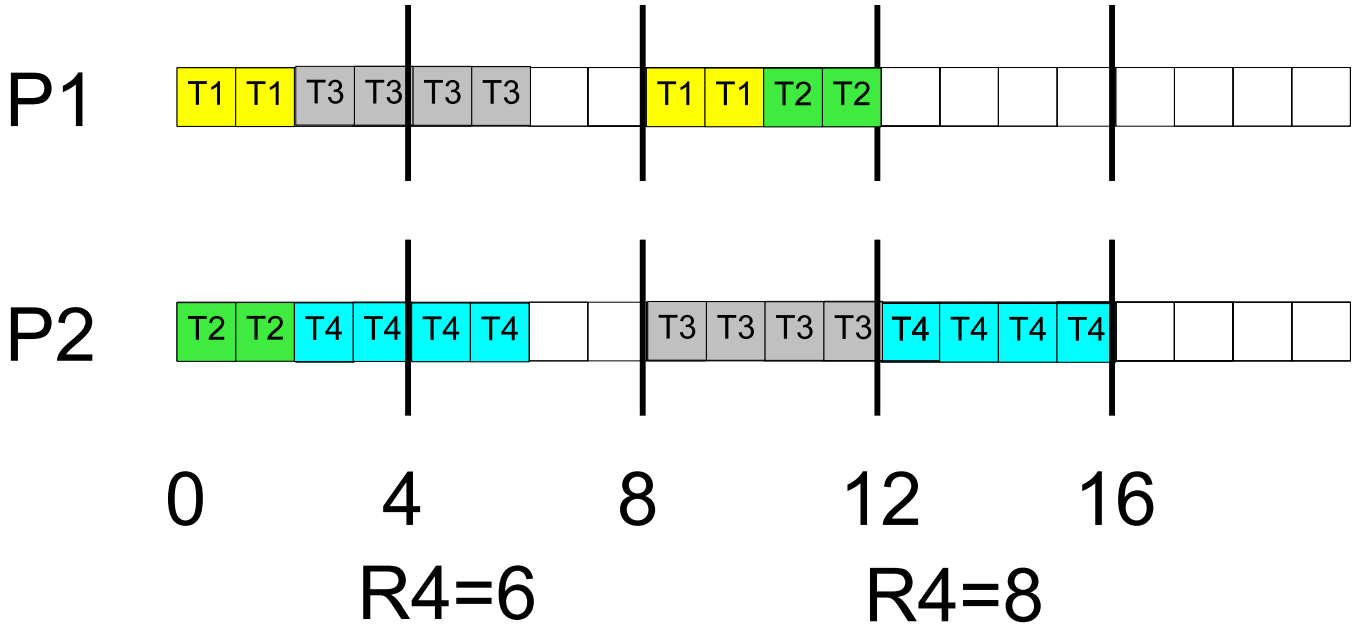
3: migration chaque unité de temps	(1,3)	(2,3)	(3,3)
2: migration travaux	(1,2)	(2,2)	(3,2)
1: migration interdite	(1,1)	(2,1)	(3,1)
	1: priorités fixes tâches	2: priorité fixes travaux	3: priorités dynamiques chaque unité de temps

- **Comparaison des algorithmes :**
  - Dominance des algorithmes à priorités dynamiques. (3,3) est la classe dominante d'algorithme. Hiérarchie différente de celle en mono-processeur : global LLF domine global EDF.
  - (1,*) sont incomparables entre eux.
  - (*,1) sont incomparables avec (*,2) et (*,3).
  - Algorithmes Pfair : avec processeurs identiques + tâches périodiques synchrones à échéances sur requêtes = algorithme optimal.

Ordonnancement global (8)

- **Instant critique :**

	Ci	Di	Pi
T1	2	2	8
T2	2	4	10
T3	4	6	8
T4	4	7	8



- En mono-processeur, l'instant critique est le pire scénario pour les tâches périodiques. Hypothèse pour le calcul du pire temps de réponse.
- N'est plus vrai en multi-processeurs.

Ordonnancement global (9)

- **Intervalle de faisabilité :**
  - En mono-processeur, l'intervalle de faisabilité permet de vérifier :
    1. Un jeu de tâches périodiques indépendantes asynchrones/synchrones, $\forall i \ : \ D_i \leq P_i$.
    2. Avec un algorithme d'ordonnancement déterministe, sans test de faisabilité !

$$[0, PPCM(\forall i : P_i) + 2 \cdot max(\forall i : S_i)]$$

  - En multi-processeurs, moins de résultats généraux :

$$[0, PPCM(\forall i : P_i)]$$

pour des jeux de tâches similaires mais synchrones uniquement, et pour des algorithme à priorités fixes seulement :-(

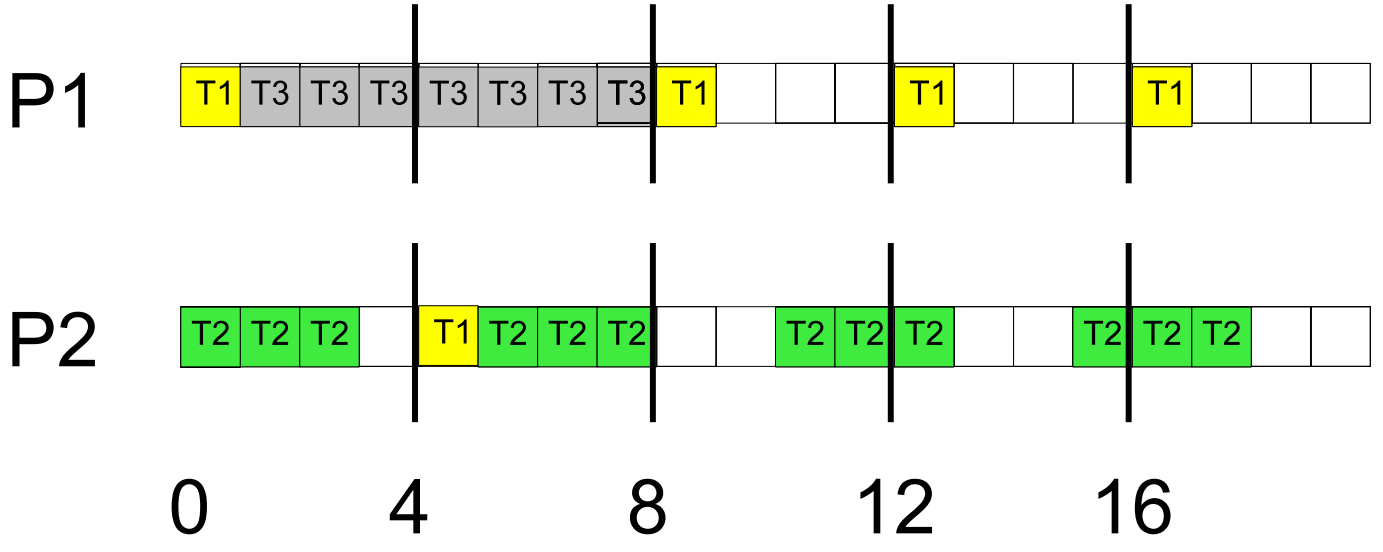
Ordonnancement global (10)

- **Anomalies d'ordonnancement :**
  - Anomalie : changement intuitivement positif de l'architecture qui conduit à un jeu de tâches non faisable.
  - Méthode de vérification utilisée en mono-processeur : vérification pire cas (ex : tâches sporadiques validées comme des tâches périodiques).
  - Paramètres concernés : C_i, P_i $\Longrightarrow$ baisse du taux d'utilisation.

Ordonnancement global (11)

- **Anomalies d'ordonnancement :**

	Ci	Pi	Di
T1	1	4	2
T2	3	5	3
T3	7	20	8

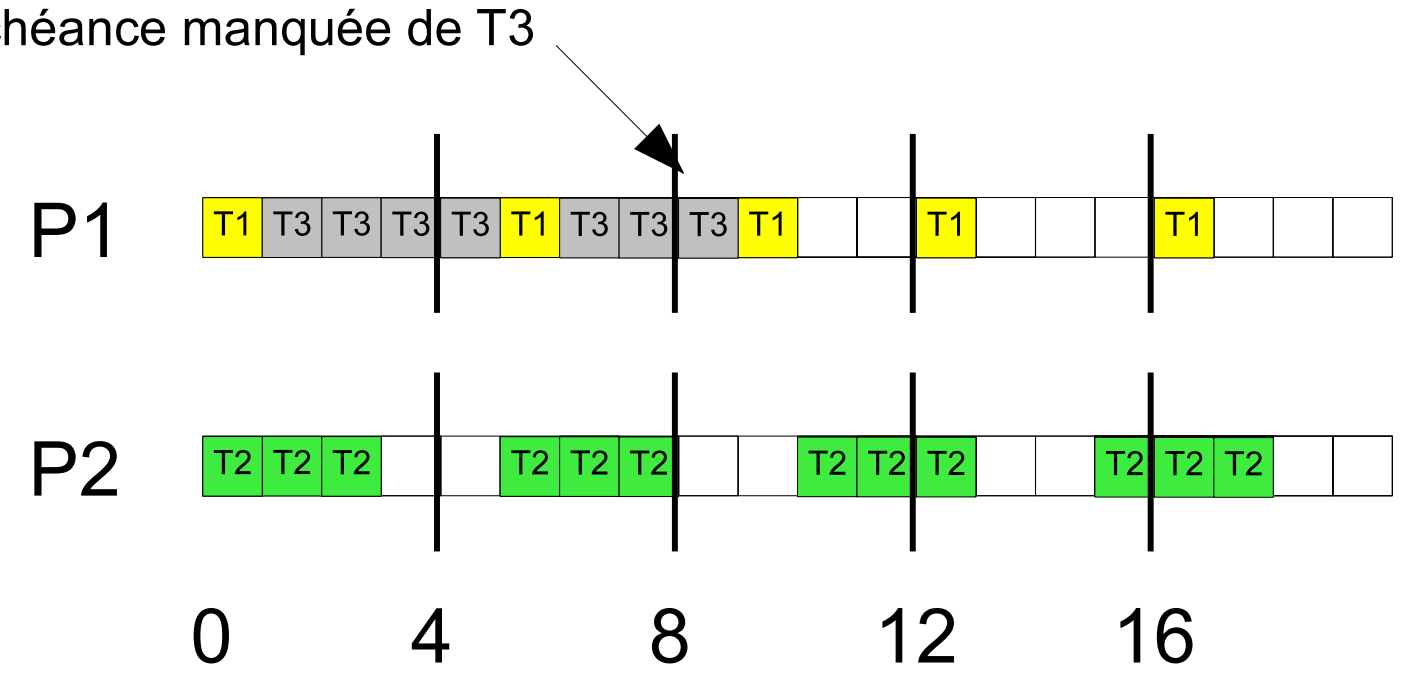


- Migration possible chaque unité de temps.
- Jeu de tâches faisable.

Ordonnancement global (12)

- **Anomalies d'ordonnancement :**

	Ci	Pi	Di
T1	1	5	2
T2	3	5	3
T3	7	20	8



- Migration possible chaque unité de temps.
- Augmentation de P_1 qui conduit à un jeu de tâches **non faisable**.

Ordonnancement global (13)

- **Principe des algorithmes Pfair :**
  - On cherche un partage équitable des processeurs (Proportionate Fair [AND 05]).
  - Algorithme optimal si processeurs identiques + tâches périodiques synchrones échéances sur requêtes.
  - Orienté échéances.
  - Migration nécessaire entre chaque unité de temps.
  - Pas de parallélisme.
  - Beaucoup de commutations de contexte.

Ordonnancement global (14)

- **Principe des algorithmes Pfair :**

1. On cherche à exécuter les tâches selon un taux constant (modèle fluide), c-à-d :

$$\forall i : WT(T_i, t) = t \cdot \frac{C_i}{P_i} \quad sur \quad [0, t)$$

2. Peut être approché par :

$$retard(T_i, t) = WT(T_i, t) - \sum_{k=0}^{t-1} Sched(T_i, k)$$

où $Sched(T_i, t) = 1$ quand T_i est ordonnancée dans intervalle $[t, t+1[$ et $Sched(T_i, t) = 0$ sinon

3. Un ordonnancement est dit Pfair si et seulement si :

$$\forall i, T_i, t : -1 \leq retard(T_i, t) \leq 1$$

4. Un ordonnancement Pfair est par construction faisable.

Ordonnancement global (15)

- **Fonctionnement des algorithmes Pfair :**
  - Découper chaque tâche en plusieurs sous-tâches, capacité d'**une** unité de temps (quantum).
  - Affectation d'une priorité dynamique $d_{(T_i,j)}$ et d'une date de réveil $r_{(T_i,j)}$ pour chaque sous-tâche :

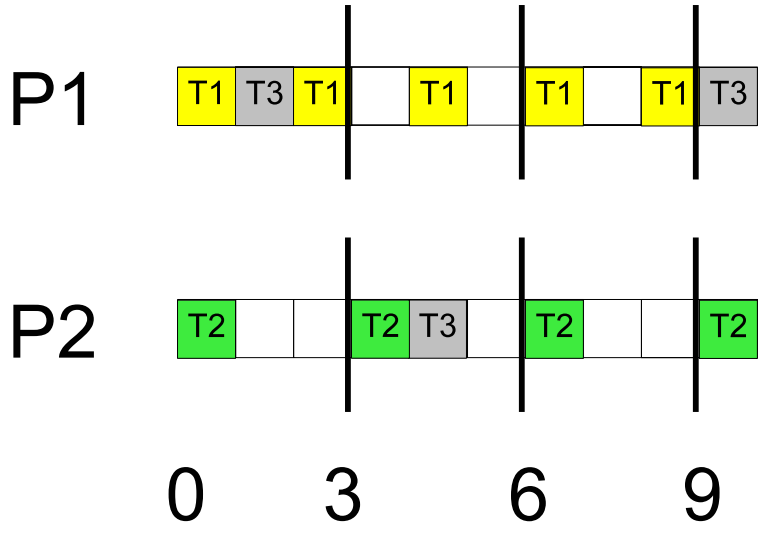
$$d_{(T_i,j)} = \left\lceil \frac{j}{C_i/P_i} \right\rceil$$

$$r_{(T_i,j)} = \left\lfloor \frac{j-1}{C_i/P_i} \right\rfloor$$

  - Ordonnancement selon $d_{(T_i,j)}$ (échéance la plus courte).
  - Variantes (optimales) si échéances identiques des sous-tâches: PF, PD, PD^2.

Ordonnancement global (16)

	Ci	Pi
T1	1	2
T2	1	3
T3	2	9



- $d_{(T_1,1)} = \left\lceil \frac{1}{1/2} \right\rceil = 2$
- $d_{(T_2,1)} = \left\lceil \frac{1}{1/3} \right\rceil = 3$
- $d_{(T_3,1)} = \left\lceil \frac{1}{2/9} \right\rceil = \lceil 4,5 \rceil = 5$
- $d_{(T_3,2)} = \left\lceil \frac{2}{2/9} \right\rceil = 9$

- $r_{(T_1,1)} = \left\lfloor \frac{1-1}{1/2} \right\rfloor = 0$
- $r_{(T_2,1)} = \left\lfloor \frac{1-1}{1/3} \right\rfloor = 0$
- $r_{(T_3,1)} = \left\lfloor \frac{1-1}{2/9} \right\rfloor = 0$
- $r_{(T_3,2)} = \left\lfloor \frac{2-1}{2/9} \right\rfloor = \lfloor 4,5 \rfloor = 4$

Ordonnancement global (17)

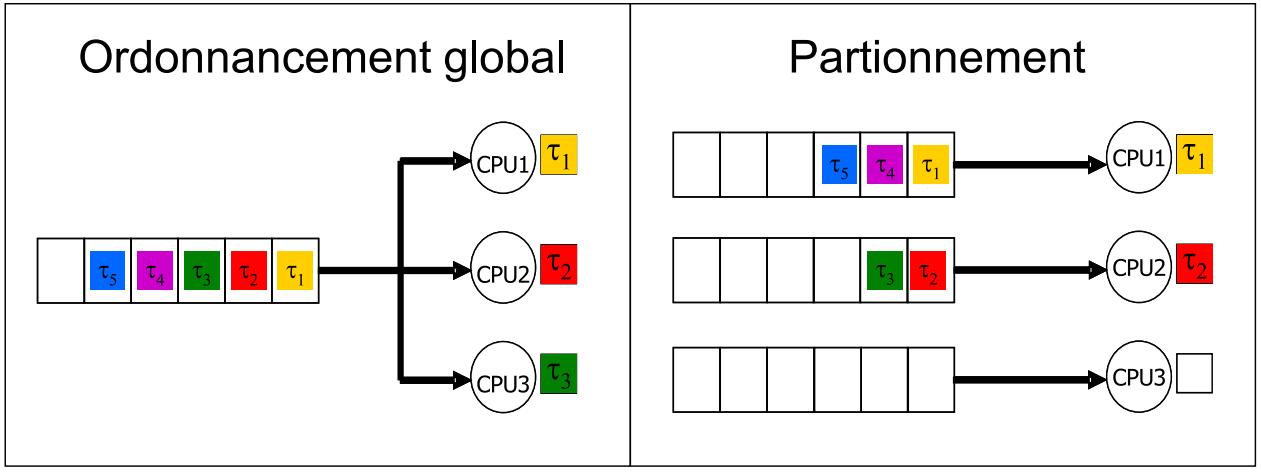
- **L'ordonnancement global est donc un domaine différent :**
  - Propriétés différentes (dominance/optimalité des algorithmes, instant critique, intervalle de faisabilité, ...).
  - Paramètres supplémentaires de l'architecture : migration, affection tâches/processeurs, types de processeurs, ...

$\Longrightarrow$ Résultats/tests de faisabilité similaires mais moins généraux.
$\Longrightarrow$ Nous nous sommes limités à des processeurs identiques, sans ressources partagées (ex : cache, bus mémoire).
$\Longrightarrow$ Nous nous sommes limités à un modèle de tâches simplifié.
$\Longrightarrow$ Nous n'avons abordé ni les dépendances entre tâches (ressources partagées, contrainte de précédences), ni les communications.
$\Longrightarrow$ Anomalies d'ordonnancement et non-sustainability.

Sommaire

1. Introduction et rappels.
2. Ordonnancement global, illustration avec les architectures multi-coeurs.
3. Ordonnancement par partitionnement, illustration avec les architectures réparties.
4. Ordonnancement par approches mixtes.
5. Résumé.
6. Acronymes.
7. Références.
8. Remerciements.

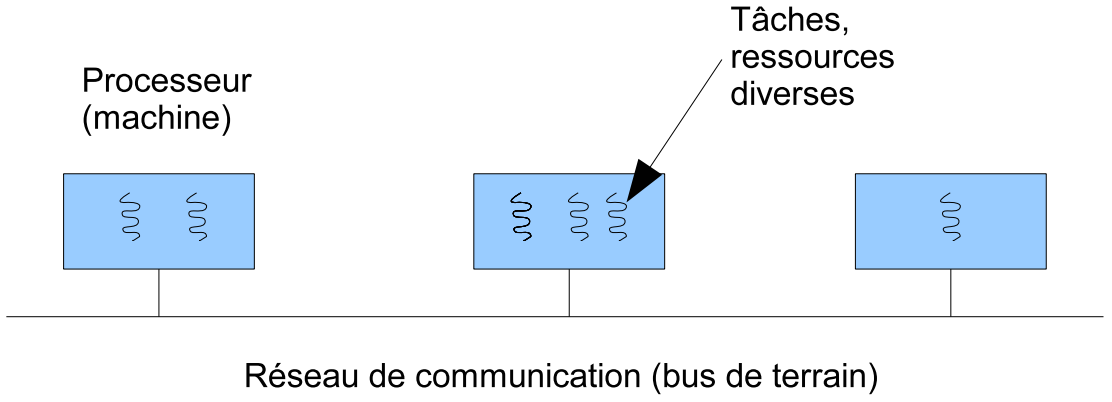
Approches d'ordonnancement



- Ordonnancement dans un système réparti/multi-processeurs :

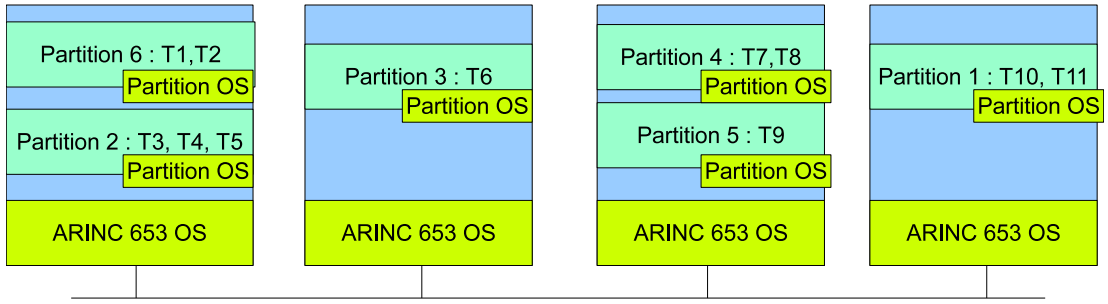
1. **Ordonnancement global :** choisir d'abord la tâche, puis placer la tâche sur un des processeurs libres. Approche en-ligne.
2. **Ordonnancement par partitionnement :** placement des tâches pour ordonnancement local, puis vérification éventuelle des délais de bout en bout. Approche hors-ligne.

Illustration : systèmes répartis (1)



1. **Conception/réalisation d'un équipement** :
   - Vérification contraintes locales. Test de la faisabilité pour les fonctions centralisées sur un processeur.
2. **Architecture/ingénierie du système :**
   - Configuration du réseau et placement des tâches.
   - Vérification des délais/contraintes de bout en bout : contraintes liant plusieurs processeurs entre eux [TIN 94, LEB 95, RIC 01, CHA 95]. Fonctions réparties sur plusieurs processeurs.

Illustration : systèmes répartis (2)



- **Exemple:** ARINC 653

1. Standard avionique. Assure le partitionnement temporel et spatial des ressources (processeur, mémoire, ...).
2. API niveau système: C, Ada, tâches, partitions, communications intra et inter partitions.
3. Associé au standard de certification DO-178B : exécuter sur la même architecture des applications de criticité différente.
4. Deux niveaux d'ordonnancement : ordonnancement des partitions vs des tâches.

Illustration : systèmes répartis (3)

1. Placement des tâches ? Paramètres des tâches (priorités) ?
2. Calcul temps acheminement des messages (accès au réseau) ?
3. Calcul délais de bout en bout émetteur/récepteur ?
4. Calcul temps de mémorisation dans les tampons. Taille des tampons de communications (ex: sockets) ?

Sommaire

- Approche par partitionnement, illustration avec les systèmes répartis :

1. Placement de tâches, paramètres des tâches.
2. Ordonnancement de messages.
3. Contraintes de bout en bout.
4. Dimensionnement de tampons.

Placement de tâches (1)

- **Comment statiquement affecter un ensemble de tâches à un ensemble de processeurs ?**
  - Problème de "bin-packing" (emballage): trouver le rangement le plus économique possible pour un ensemble d'articles dans des boîtes.
  - NP-difficile $\Longrightarrow$ heuristiques de partitionnement.
- **Exemples de paramètre :**
  - Processeurs (identiques ou non), tâches (traitements regroupés, priorités, périodes, capacités).
  - Prendre en compte les communications entre les tâches, les ressources partagées ...
- **Exemples de fonction objective:**
  - Minimiser le nombres de processeurs, les communications, les latences, ...
- **Difficultés :**
  - Comment comparer/évaluer l'efficacité des heuristiques?
  - Paramètres nombreux et souvent dépendants entre eux.
  - Fonctions objectives contradictoires. Contraintes techniques.

Placement de tâches (2)

- **Exemples d'heuristiques de placement (mono-objectif) :**

  - Très souvent basées sur RM [OH 93].

  - Principales heuristiques d'ordonnancement : Rate-Monotonic First-Fit, Next-Fit ou Best-Fit [DHA 78, OH 93].

  - Autres : RM-FFDU[OH 95], RM-ST, RM-GT et RM-GT/M [BUR 94], RM-Matching[DAV 09], EDF-FFD (First Fit Decreasing) [BAR 03], EDF-FF/EDF-BF, FFDUF[DAV 09].

Placement de tâches (3)

- **Exemple de Rate Monotonic Next Fit** :

1. Les tâches sont classées dans l'ordre croissant de leur taux d'utilisation/période. On commence par la tâche $i = 0$ et le processeur $j = 0$.
2. On assigne la tâche i au processeur j si la condition de faisabilité est respectée (ex: $U \leq 0.69$).
3. Dans le cas contraire, on assigne la tâche i au processeur $j + 1$.
4. On passe à la tâche suivante.
5. Arrêt lorsqu'il n'y a plus de tâche à ordonnancer, j = nombre de processeurs nécessaires.

Placement de tâches (4)

- Problème de Next Fit = taux d'utilisation des processeurs parfois bas. Autres heuristiques similaires:

  - First Fit : pour chaque tâche i, regarder les processeurs en commençant par $j = 0$ et placer la tâche i sur le premier processeur dont le test de faisabilité est vérifié.
  - Best Fit : pour chaque tâche i, regarder tous les processeurs et placer la tâche i sur le processeur dont le test de faisabilité est vérifié et qui maximise le taux d'occupation du processeur.
  - ...

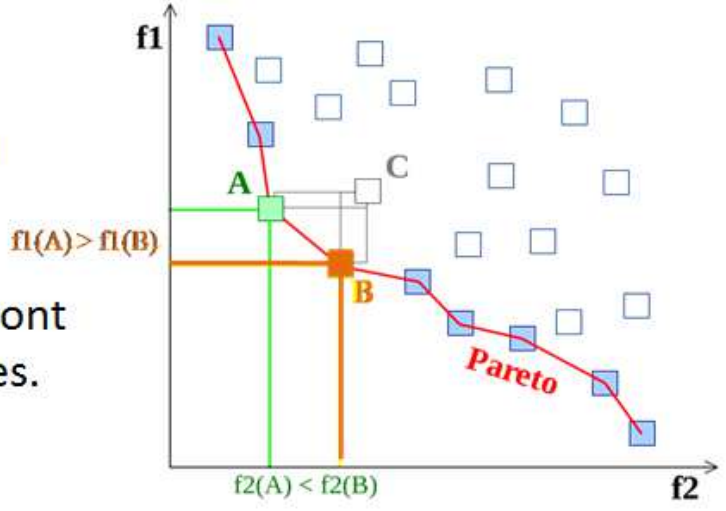
Placement de tâches (5)

- **PAES : une méta heuristique, itérative, évolutionnaire, avec plusieurs fonctions objectives contradictoires**

1. **Mutation** d'une solution pour générer un nouveau candidat : petite modification architecturale, solution voisine
2. **Evaluer** la solution mutée selon les fonctions objectives
3. **Modifier** l'archive, i.e. l'ensemble des solutions non dominées
4. **Sélectionner** une nouvelle solution pour l'itération suivante

- **Front de Pareto :** ensemble des solutions non dominées.

- Solutions A et B dominent solution C car elles sont meilleurs que C pour les deux fonctions objectives.



Paramètres des tâches (1)

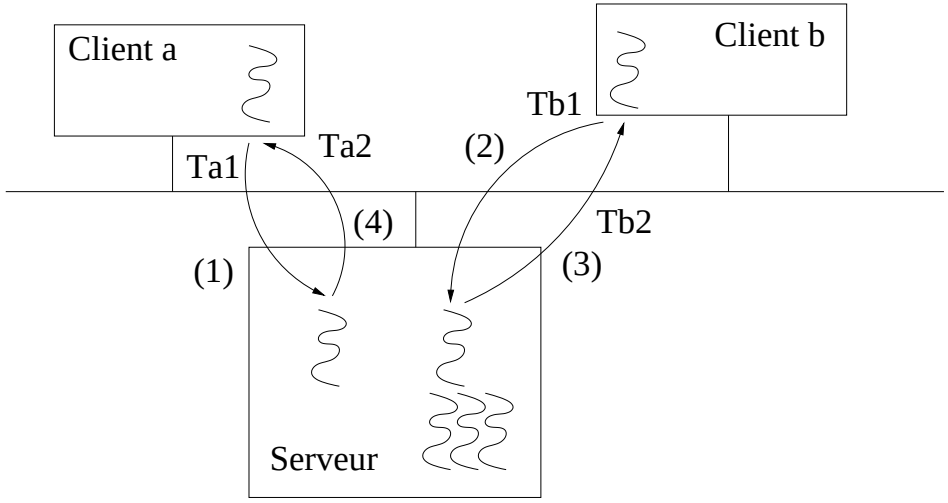
- Proposer un partitionnement (ex: soluation candidate avec PAES), implique une **affectation globale** de différents paramètres, dont la priorité des tâches :

$\Longrightarrow$ inversion de priorités.

$\Longrightarrow$ niveaux de priorité offerts sur chaque processeur (caractéristiques du système d'exploitation).

Paramètres des tâches (2)

- **Exemple :** application client-serveur.



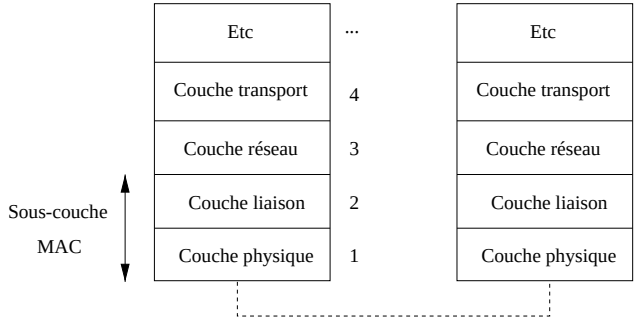
- Un serveur est une ressource partagée $\Longrightarrow$ inversion de priorité entre les invocations T_{a1}/T_{a2} et T_{b1}/T_{b2}.
- Solution : comme PCP/PIP, affectation de priorités afin de garantir l'absence d'interruption.
- Heuristique (conjointe) de partitionnement et d'affectation de priorités. Boites noires, composants sur étagère ?

Sommaire

- Approche par partitionnement, illustration avec les systèmes répartis :

1. Placement de tâches, paramètres des tâches.
2. Ordonnancement de messages.
3. Contraintes de bout en bout.
4. Dimensionnement de tampons.

Ordonnancement de messages (1)



- Transport : application, contraintes de temps.
- Réseau : commutation, routage.
- Physique/Liaison/MAC : partage du medium $\Longrightarrow$ **protocole d'arbitrage**.

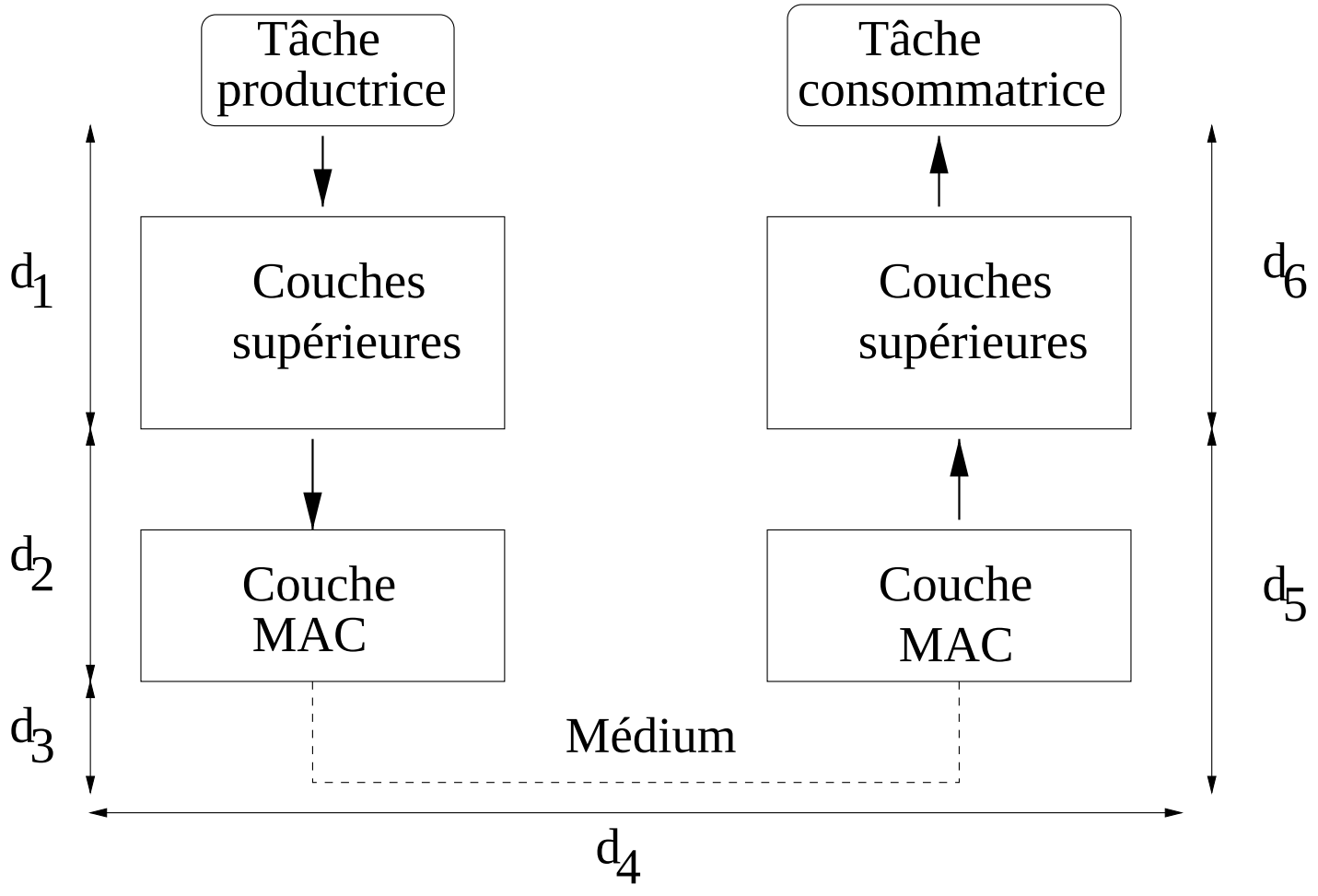
- Bus/réseaux temps réel = MAC + couche applicative/transport.
- Couche applicative adaptée (événements, variables périodiques),
- Débit faible, petite taille, contraintes environnement.

Ordonnancement de messages (2)

- **Exemples de réseaux déterministes :**
  - Productique et robotique : MAP [DWY 91], Token bus/IEEE 802.4 [PUJ 95], FIP [CIA 99], Profibus [STR 96].
  - Aéronautique/spatial/naval : DigiBus, STD MIL 1553, ARINC 429 et 629, AFDX, Spacewire. Aviation civile et militaire, la sonde PathFinder, systèmes navals [DEC 96, HEA 96].
  - Industrie automobile : bus CAN [ZEL 96].
  - Domotique : BatiBus[CIA 99].

Ordonnancement de messages (3)

- Temps de communication composé de [COT 00] :



Ordonnancement de messages (4)

- Temps de communication composé de [COT 00] :

  - d_1 et d_6 = délais de traversée des couches ; d_5 = délai de réception. Calcul facile, peu variable et borné.
  - d_3 = délai de transmission sur le médium. Variable, calcul facile $\Longrightarrow$ taille message/débit.
  - d_4 = délai de propagation. Variable, calcul facile $\Longrightarrow$ taille réseau/vitesse.
  - d_2 **= délai d'attente pour l'accès au réseau. Variable, dépend du protocole d'arbitrage.**

Ordonnancement de messages (5)

- L'arbitrage définit la méthode d'accès au médium.

Objectif principal : comportement prédictible des temps accès.

- Eléments de taxinomie des protocoles d'arbitrage :
  - Algorithme par coopération ou compétition.
  - Arbitrage symétrique ou asymétrique. Arbitrage centralisé/réparti. Notion de maîtres, d'arbitres et d'esclaves : qui prend l'initiative de la communication ?
  - Synchones ou asynchones $\Longrightarrow$ y a t il une horloge globale à tous les coupleurs ?
  - Topologies: bus, réseaux étoile, lien point à point.

Ordonnancement de messages (6)

- Principaux protocoles d'arbitrage[UPE 94] :
  - **Mono-maître :** ARINC 429, DigiBus
    - 1 émetteur et n récepteurs. Bus mono-directionnel.
    - n bus si communications bi-directionnelles.
  - **CSMA/CA :** CAN, Batibus
    - Une priorité fixe est associée à chaque station.
    - Chaque station émet quand elle le souhaite.
    - Les éventuelles collisions sont réglées par le biais de la priorité : la station de plus forte priorité obtient le médium.
  - **Protocole à jeton :** Token-Bus, Profibus
    - Une jeton circule selon un ordre fixe entre les stations (ex : topologie en boucle).
    - Une station reçoit le jeton émet ses données et transmet le jeton à la station suivante.

Ordonnancement de messages (7)

- **Polling :** FIP, MIL-STD-1553
  - Une station dédiée, dite "arbitre", émet un message sur le médium, invitant un "maître" à émettre.
  - En réponse, le maître émet ses données. Puis, l'arbitre interroge le maître suivant.
  - L'arbitre dispose d'une liste d'invitations fixe qu'il parcourt de façon séquentielle.
- **TDMA [KOO 95] :** ARINC 629
  - Une station dédiée, appelée "arbitre" émet cycliquement une trame de synchronisation (timers).
  - Chaque maître émet alors, à un instant relatif par rapport à la trame de synchronisation.
  - L'ordre d'émission des maîtres est prédéterminé et fixe $\Longrightarrow$ partage temporel du médium.

Ordonnancement de messages (8)

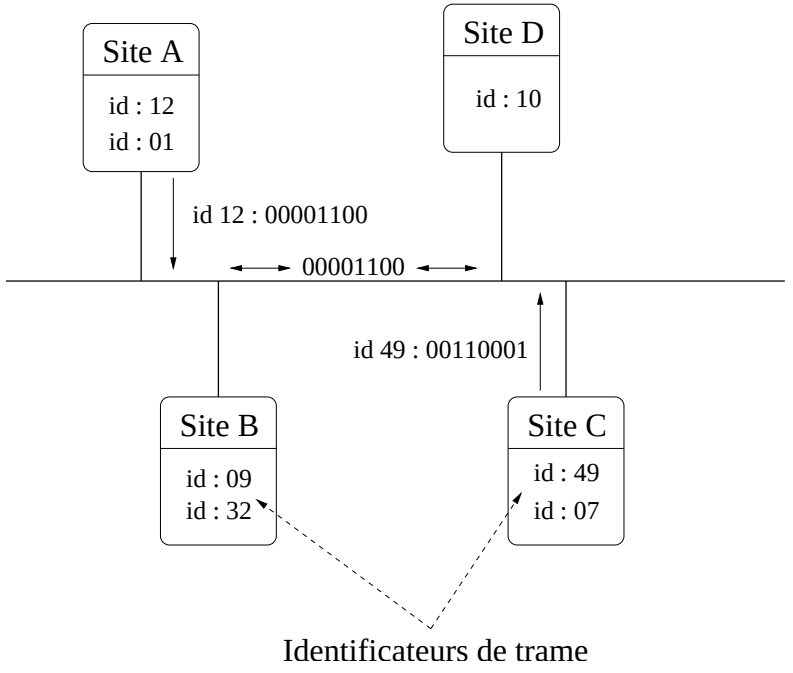
- **CSMA/CA : le bus CAN**
  - Créé par Bosh et Intel pour les applications automobiles.
  - Adopté aujourd'hui par de nombreux constructeurs automobiles et dans divers domaines applicatifs.
  - Nombreux standards.
  - Réseau multi-maîtres symétriques.
  - Transmission par diffusion.
  - Topologie en bus, paires torsadées, généralement longueur maximum 40 m pour un débit de 1 MBit/s.
  - Composants très fiables à faibles coût (automobile).

Ordonnancement de messages (9)

- Identificateurs de trame = priorités. Identificateurs uniques et émis par une seule station.
- Arbitrage CAN = arbitrage par compétition = CSMA non destructif (CSMA/CA) :

  - Lorsque le bus est libre, émission bit à bit de l'identificateur avec écoute de la porteuse.
  - Un bit à 1 (récessif) est masqué par un bit à 0 (dominant).
  - Tout coupleur lisant un bit différent de celui qu'il vient d'émettre passe en réception. Puis, réémet immédiatement lorsque la porteuse est de nouveau libre.
  - Émission bit à bit + écoute porteuse = faible débit/taille du réseau.

Ordonnancement de messages (10)



- A, B, C et D sont des maîtres (émetteurs). Chacun détient une liste d'identificateurs uniques.
- A et C commencent à émettre en même temps les identificateurs 12 et 49 (priorités).
- Lors de la transmission du 3 ème bit, le site C passe en réception.
- Le site A gagne et transmet son information sur le bus (de 0 et 8 octets).

Ordonnancement de messages (11)

- De l'ordonnancement de tâches aux messages :

Ordo. de tâches	Ordo. de messages
Tâches	Messages
Processeur	Médium de communication
Capacité	Temps de transmission + temps de propagation + temps de traversée
Temps d'interférence	Temps d'accès
Temps de réponse	Temps de communication de tâche à tâche

- Points communs : échéance, période, etc.
- Spécificité : caractère **non préemptif**.

Ordonnancement de messages (12)

- Comment calculer le temps de communication d'un message périodique ? En appliquant les mêmes méthodes que pour le partage du processeur.

- Analyse avec le bus CAN :

  - Identificateur d'un message CAN = priorité fixe d'une tâche. Affecter les identificateurs selon Rate Monotonic.
  - Le protocole d'arbitrage de CAN est un ordonnanceur à priorité fixe *non préemptif.* Temps de réponse dans le cas non préemptif [GEO 96]:

$$r_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{r_i}{P_j} \right\rceil C_j + max(C_k, \forall k \in lp(i))$$

avec $lp(i)$, l'ensemble des tâches de plus faible priorité que i.

Ordonnancement de messages (13)

- Exemple :

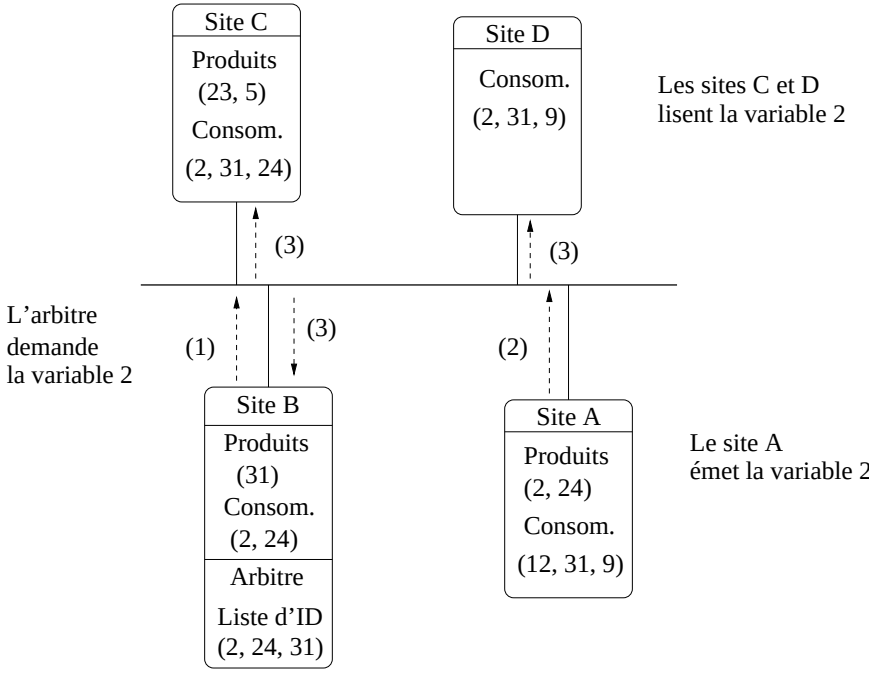
Msg	Période (en ms)	Transmi./Propag. (en μs)	Accès (en μs)	Total (en μs)
a	4	80	150	230
b	16	150	410	560
c	4	150	0	150
d	8	180	230	410
e	64	250	560	810

- Temps de traversée des couches et de réception supposés nuls.
- Calculs par Cheddar, un outil d'ordonnancement pour des tâches essentiellement périodiques.

Ordonnancement de messages (14)

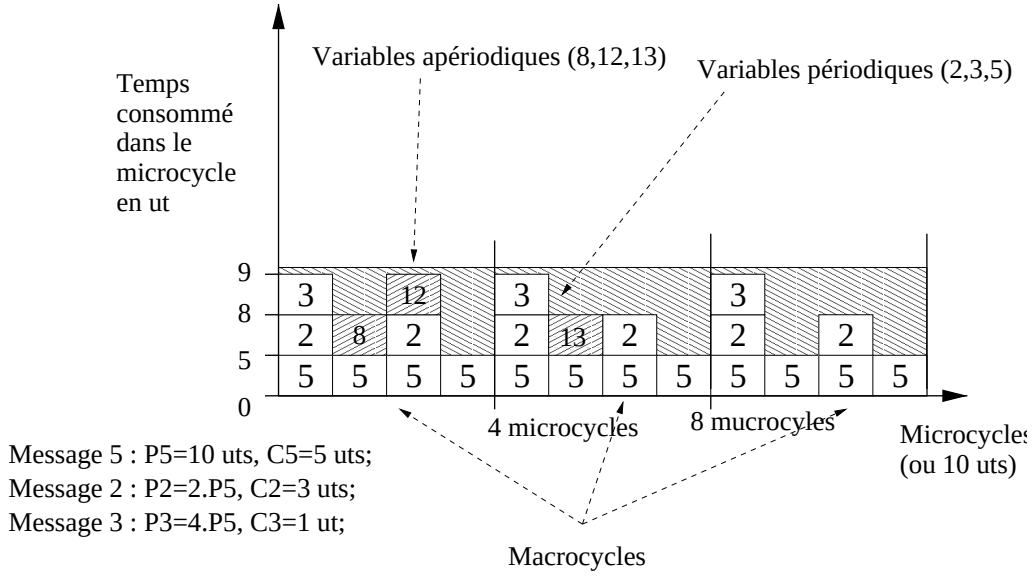
- **Polling : le bus FIP**
  - Réseau pour la productique.
  - Polling : multi-maîtres, arbitre centralisé.
  - Topologie en bus ou en étoile. Paire torsadée. Longueur de 4000 m avec un débit de 1Mbit/s.
  - Sûreté de fonctionnement : coupleur bi-médium.
  - Orienté variables (de 1 à 128 octets). Variable = valeur de capteur.
  - Protocole de type producteurs/consommateurs.
  - Diffusion de variables sur le bus.

Ordonnancement de messages (15)



- L'arbitre détient une liste de variables qu'il exploite séquentiellement :
  - L'arbitre indique la variable à transmettre par un message (1).
  - Le producteur émet la valeur sur le bus (2).
  - Les consommateurs lisent le bus (3).

Ordonnancement de messages (16)



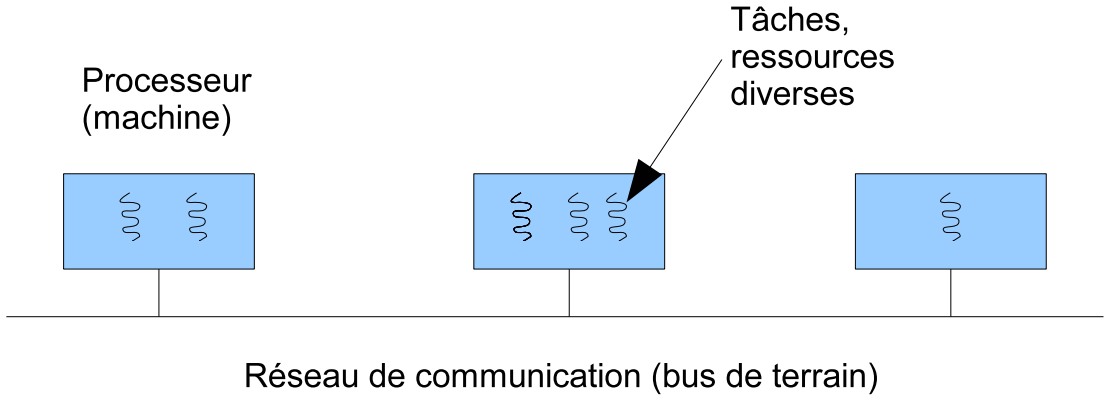
- **Table d'arbitrage :**
  - Notion de microcycle (plus petite période) et de macrocycle (PPCM des périodes).
  - L'arbitre est une tâche périodique, qui se réveille à chaque début de microcycle pour la transmission des variables périodiques d'abord, puis, s'il reste du temps, des variables apériodiques.

Sommaire

- Approche par partitionnement, illustration avec les systèmes répartis :

1. Placement de tâches, paramètres des tâches.
2. Contraintes de précédence.
3. Ordonnancement de messages.
4. Contraintes de bout en bout.
5. Dimensionnement de tampons.

Faisabilité dans le cas réparti (1)



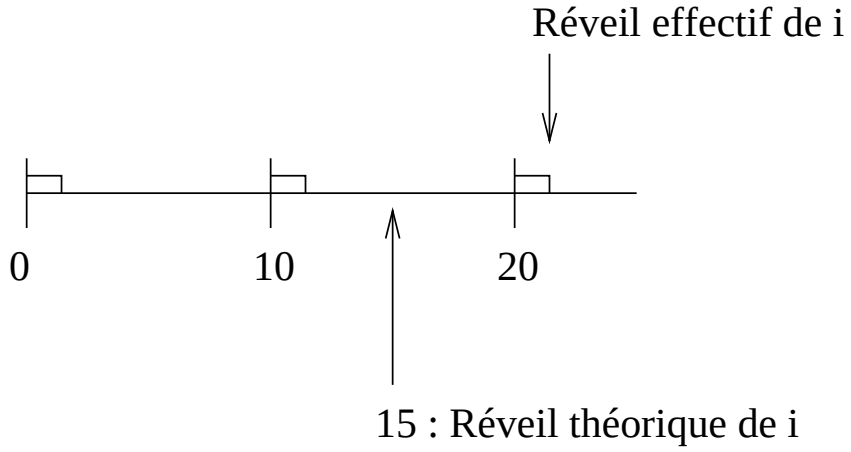
1. **Conception/réalisation d'un équipement** :
   - Vérification contraintes locales. Test de la faisabilité pour les fonctions centralisées sur un processeur.
2. **Architecture/ingénierie du système :**
   - Configuration du réseau et placement des tâches.
   - Vérification des délais/contraintes de bout en bout : contraintes liant plusieurs processeurs entre eux [TIN 94, LEB 95, RIC 01, CHA 95]. Fonctions réparties sur plusieurs processeurs.

Faisabilité dans le cas réparti (2)

- Plusieurs approches de vérification possibles :
  - Approche basée sur Chetto et Blazewicz [CHE 90].
  - Approche basée sur les offsets [BAT 97, PAL 03].
  - Approche Holistique [TIN 94, LEB 95, RIC 01, CHA 95].

Un modèle de retard : la gigue (1)

- Utilisation de la gigue, ou **Jitter**. Exemple historique $\Longrightarrow$ le timer d'un système est modélisé comme une tâche périodique avec $P_{timer} = 10\ ms, C_{timer} = 3\ ms$.
- On souhaite réveiller une tâche i à l'instant $t = 15\ ms$.



Date effective de réveil de la tâche i = $23\ ms$. Gigue $J_i = 8\ ms$.

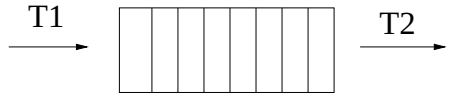
- Pire temps de réponse :

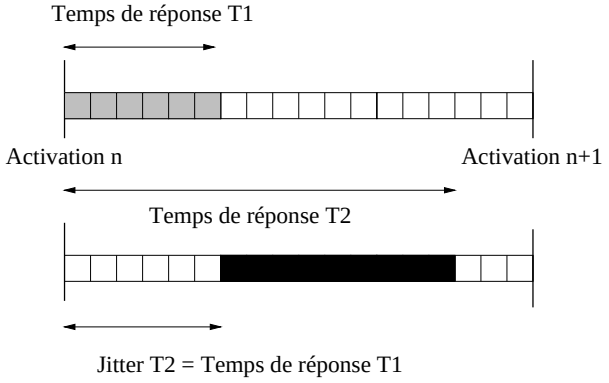
$$r_i = w_i + J_i$$

$$w_i = C_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{w_i + J_j}{P_j} \right\rceil C_j$$

Un modèle de retard : la gigue (2)

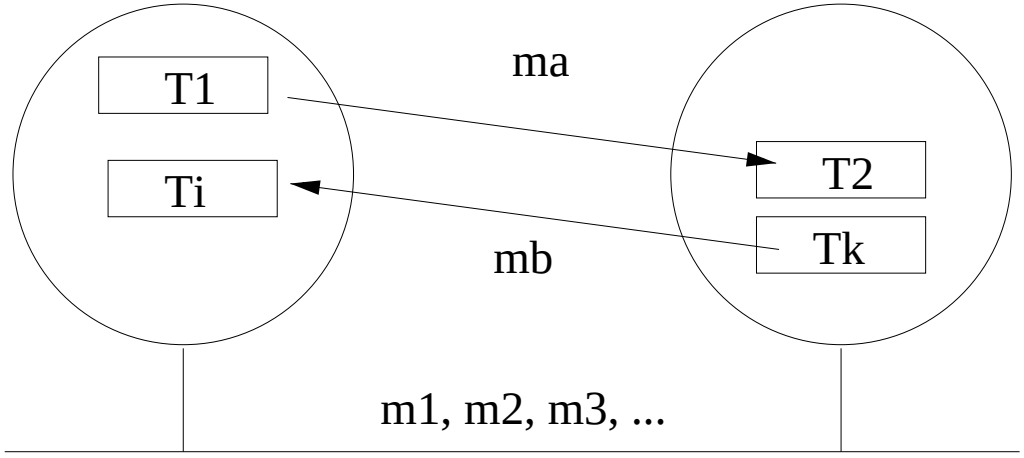
T1 : C1=6 ; P1=18 (gris)
T2 : C2=9 ; P2=18 (noir)





- Exemple du producteur/consommateur :
  - T1 et T2 sont activées toutes les 18 unités de temps.
  - T1 lit un capteur et transmet la valeur vers T2 qui l'affiche à l'écran.
  - T2 doit être activée sur terminaison de T1.
  - Quel est le temps de réponse de T2 ?

Temps de réponse bout en bout (1)



- **Contrainte à vérifier :** $r_{(T1+ma+T2)} \leq D$
- **Le calcul Holistique :** injection du temps de réponse sous la forme d'une gigue[TIN 94] :

  - Soit r_{T1}, le temps de réponse de T1.
  - r_{ma} est calculé, en fonction du protocole d'arbitrage, des autres messages périodiques et tel que $J_{ma} = r_{T1}$.
  - r_{T2} est calculé avec $J_2 = r_{ma}$.
  - **Calcul itératif jusqu'à convergence.**

Temps de réponse bout en bout (2)

```
10    ∀i : J_i := 0, r_i := 0, r'_i := 0;
20    ∀i : Calculer_temps_de_réponse (r_i);
30    Tant que (∃i : r_i ≠ r'_i) {
40        ∀i : J_i := max(J_i, ∀j avec j ≺ i : r_j);
50        ∀i : r'_i := r_i;
60        ∀i : Calculer_temps_de_réponse (r_i);
70    }
```

Si i est une tâche :

$$r_i = J_i + w_i \;\; avec \;\; w_i = C_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{w_i + J_j}{P_j} \right\rceil C_j$$

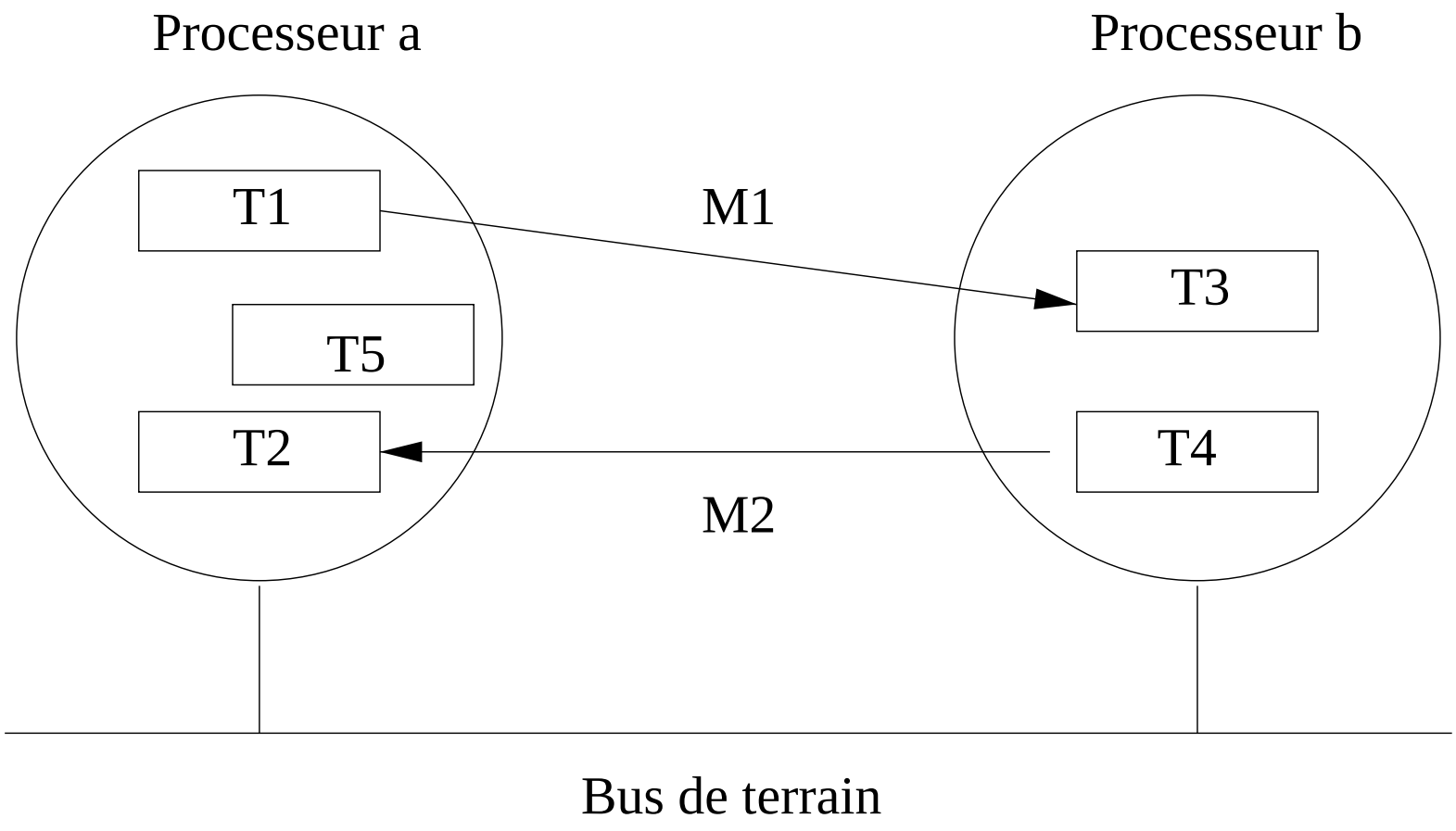
Si i est un message :

$$r_i = J_i + M_i$$

dont le temps de communication de tâche à tâche est borné par M_i.

Temps de réponse bout en bout (3)

- Soit le placement suivant :



Temps de réponse bout en bout (4)

- Et les paramètres suivants :

Tâche	Période	Capacité	Priorité	Processeur
T_1	100	4	1	a
T_2	60	5	2	a
T_3	100	3	2	b
T_4	60	2	1	b
T_5	90	3	3	a

Message	Période	Délai de communication de tâche à tâche
M_1	100	6
M_2	60	1

Temps de réponse bout en bout (5)

- **Lignes 10-20 :** $\forall i : J_i = 0$

Message/Tâche	M_1	M_2	T_1	T_2	T_3	T_4	T_5
J_i	0	0	0	0	0	0	0
r_i	6	1	4	9	5	2	12

- **Première itération, lignes 40-60 :** modification jitter + calcul temps de réponse. $J_{M_1} = r_{T_1}$, $J_{M_2} = r_{T_4}$, $J_{T_3} = r_{M_1}$ et $J_{T_2} = r_{M_2}$.

Message/Tâche	M_1	M_2	T_1	T_2	T_3	T_4	T_5
J_i	**4**	**2**	0	**1**	**6**	0	0
r_i	**6+4 =10**	**1+2 =3**	4	**9+1 =10**	**5+6 =11**	2	12

Temps de réponse bout en bout (6)

- **Deuxième itération, lignes 40-60 :** modification jitter + calcul temps de réponse. $J_{M_1} = r_{T_1}$, $J_{M_2} = r_{T_4}$, $J_{T_3} = r_{M_1}$ et $J_{T_2} = r_{M_2}$.

Message/Tâche	M_1	M_2	T_1	T_2	T_3	T_4	T_5
J_i	**4**	**2**	0	**3**	**10**	0	0
r_i	**6+4 =10**	**1+2 =3**	4	**9+3 =12**	**5+10 =15**	2	12

- **Troisième itération, lignes 40-60 :** modification jitter $J_{M_1} = r_{T_1}$, $J_{M_2} = r_{T_4}$, $J_{T_3} = r_{M_1}$ et $J_{T_2} = r_{M_2}$. Jitters identiques => même temps de réponse = convergence et arrêt des calculs.

Message/Tâche	M_1	M_2	T_1	T_2	T_3	T_4	T_5
J_i	**4**	**2**	0	**3**	**10**	0	0

Sommaire

- Approche par partitionnement, illustration avec les systèmes répartis :

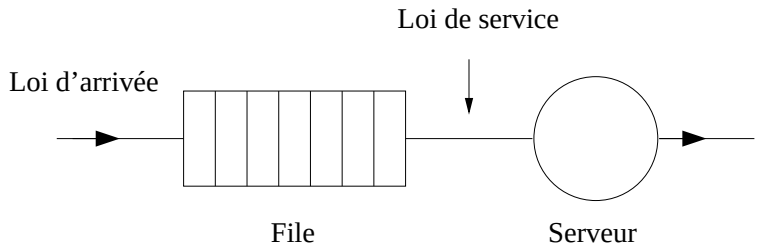
1. Placement de tâches, paramètres des tâches.
2. Ordonnancement de messages.
3. Contraintes de bout en bout.
4. Dimensionnement de tampons.

Dimensionnement des tampons

- Problèmes soulevés :

1. Déterminer la taille des tampons afin d'éviter un éventuel débordement, et dans certain cas une famine.
2. Déterminer le pire délai de mémorisation d'un message $\Longrightarrow$ calcul holistique.

Dimensionnement de tampon (1)



- **Objectif :** description loi arrivée λ et loi service μ.
- Notation de kendall : $\lambda/\mu/n$:
  - λ : loi d'arrivée des clients (M,G,D).
  - μ : loi de service des clients (M,G,D).
  - n : nombre de serveurs.

- Exemples : $M/M/1$, $M/D/1$, $M/G/1$, $P/P/1$, ...

Dimensionnement de tampon (2)

- **Critères de performance recherchés :** L=nombre moyen de message, W=délai de mémorisation moyen d'un message dans la file d'attente (W_q et L_q), ou dans le serveur (L_s et W_s). Avec :

$$W = W_q + W_s$$

$$L = L_q + L_s$$

- **Loi de Little :** relation entre délai/latence et taille/dimensionnement.

$$L = \lambda . W$$

$$W = \frac{L}{\lambda}$$

Dimensionnement de tampon (3)

- **Critères de performance des principaux modèles :**

	L	L_q	W	W_q
M/M/1	$\frac{\lambda W_s}{1-\rho}$	$\frac{\lambda^2 W_s^2}{1-\rho}$	$\frac{W_s}{1-\rho}$	$\frac{\lambda W_s^2}{1-\rho}$
M/G/1	$\lambda W_s + \frac{\lambda^2(W_s^2+\sigma^2)}{2(1-\rho)}$	$\frac{\lambda^2(W_s^2+\sigma^2)}{2(1-\rho)}$	$W_s + \frac{\lambda(W_s^2+\sigma^2)}{2(1-\rho)}$	$\frac{\lambda(W_s^2+\sigma^2)}{2(1-\rho)}$
M/D/1	$\lambda W_s + \frac{\lambda^2 W_s^2}{2(1-\rho)}$	$\frac{\lambda^2 W_s^2}{2(1-\rho)}$	$W_s + \frac{\lambda W_s^2}{2(1-\rho)}$	$\frac{\lambda W_s^2}{2(1-\rho)}$

- Ces modèles ne sont pas adaptés au modèle périodique de tâches de la théorie de l'ordonnancement temps réel : réveils périodiques, service lié à l'ordonnancement déterministe, ...

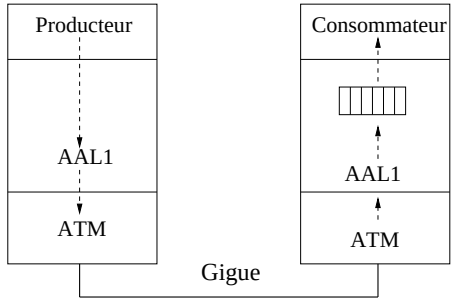
Dimensionnement de tampon (4)

- **File d'attente P/P/1[LEG 03] :**
  - "P" pour périodique.
  - n tâches périodiques producteurs et 1 tâche consommateur.
  - Contraintes temporelles des tâches respectées et $\forall i : D_i \leq P_i$.
  - Loi de conservation du débit : $\frac{\lambda}{\mu} \leq 1$.
  - 1 message produit ou consommé par activation.

- **Test de faisabilité :**

Harmonique	Non harmonique
$L_{max} = 2.n$	$L_{max} = 2.n + 1$
$W_{max} = 2.n.P_{cons}$	$W_{max} = (2.n + 1).P_{cons}$

Dimensionnement de tampon (5)



- **Résultat dans le cas ATM/AAL1 :**

1. Taille maximum des tampons régulant le trafic :

$$L_{max} = \left\lceil \frac{W_{max}}{d} \right\rceil$$

où $W_{max} = 2.gigue$. est **le délai de mémorisation d'une cellule** avec $gigue = delai_{max} - delai_{min}$.

2. C'est une illustration de la loi de Little : $L_{max} = \lambda_{max}.W_{max}$ avec $\lambda_{max} = 1/d$

Dimensionnement de tampon (6)

- **Application de la méthode AAL1/Little :**

1. **Délai de mémorisation :**

$$W_{max} = (y+1).P_{cons} + D_{cons}$$

y messages déjà présents dans le tampon. $cons$ = tâche consommateur.

2. **Borne nombre de message :**

$$L_{max} = \max_{\forall y \geq 0} \left(\sum_{prod \in PROD} \left\lceil \frac{W_{max} + O_{prod}}{P_{prod}} \right\rceil - y \right)$$

O_{prod}=désynchronisation d'un producteur vis-à-vis du consommateur.

3. **Obtention** de L_{max} par études aux limites de y. Obtention de W_{max} par application de Little.

Sommaire

1. Introduction et rappels.
2. Ordonnancement global, illustration avec les architectures multi-coeurs.
3. Ordonnancement par partitionnement, illustration avec les architectures réparties.
4. Ordonnancement par approches mixtes.
5. Résumé.
6. Acronymes.
7. Références.
8. Remerciements.

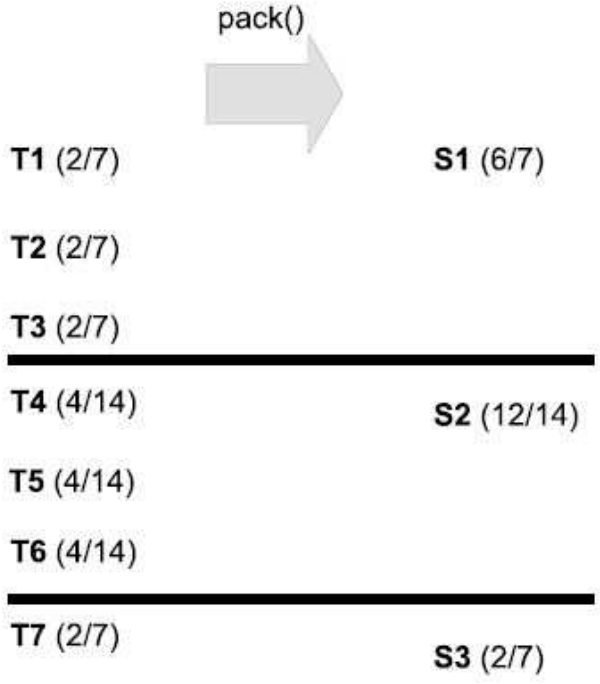
Approches mixtes (1)

- Algorithmes comportant une partie hors-ligne et une partie en ligne.
- **Exemple :** RUN (Reduction to Uniprocessor)
  - Tâches périodiques synchrones à échéances sur requêtes.
  - Optimal, réduit le nombre de préemption (vs PFair).
- **Hors ligne :** réduction, de proche en proche
  - De l'ensemble de tâches en serveurs ordonnancés sur des processeurs virtuels.
  - Jusqu'à obtenir un ordonnancement monoprocesseur.
  - Arbre de réduction.
- **En ligne :** ordonnancement en ligne sur chaque processeur (virtuel) selon EDF par lecture de l'arbre de réduction.

Approches mixtes (2)

- Hors ligne : réduction = appliquer (dual + pack) itérativement.

- **Pack :** grouper les tâches (serveurs), selon une politique de partitionnement (ex: First Fit) tel que $U \leq 1$.



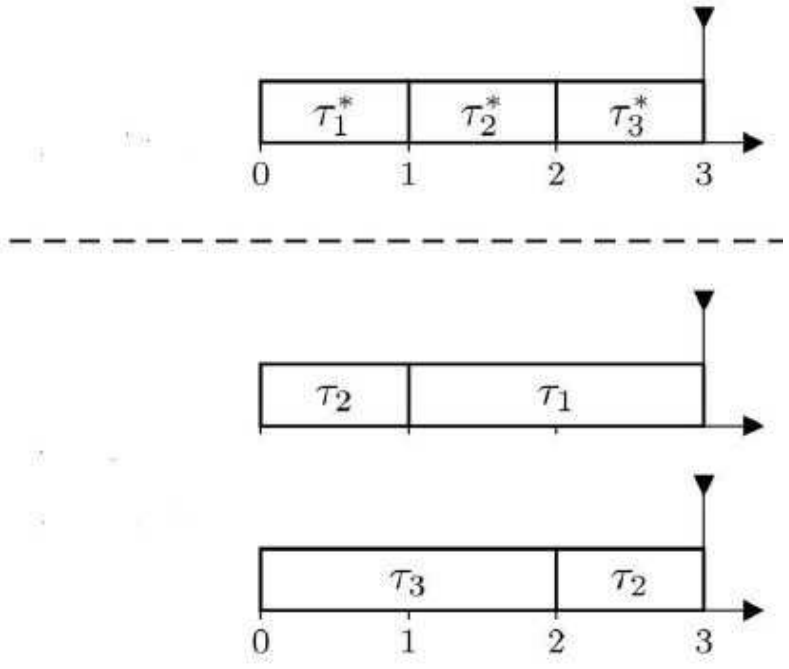
Approches mixtes (3)

- Hors ligne : réduction = appliquer (dual + pack) itérativement.

- **Dual :** calcul de l'ordonnancement d'un groupe de tâches (duales) sur un nombre inférieur de processeur, afin de calculer l'ordonnancement équivalent de ses tâches (primales).

- Equivalence d'un ordonnancement de jeu de tâche dual et primal.

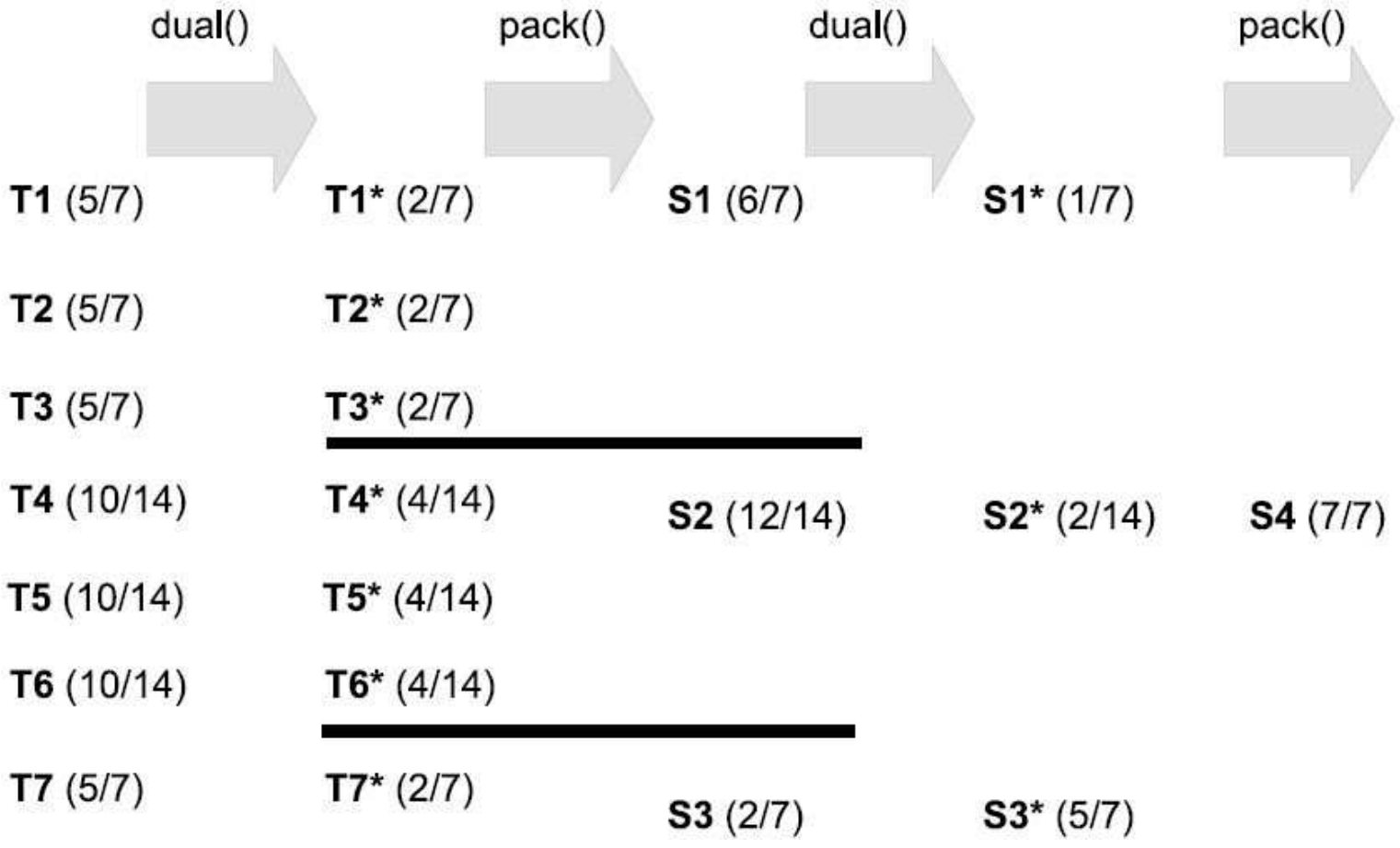
Approches mixtes (4)



- Soit une tâche T_i définie par $(C_i, D_i = P_i)$. T_i* est duale de T_i quand :
  - Les échéances de T_i^* sont identiques à celles de T_i.
  - $U_i^* = 1 - U_i$.

Approches mixtes (5)

- Hors ligne : réduction = appliquer (dual + pack) itérativement.

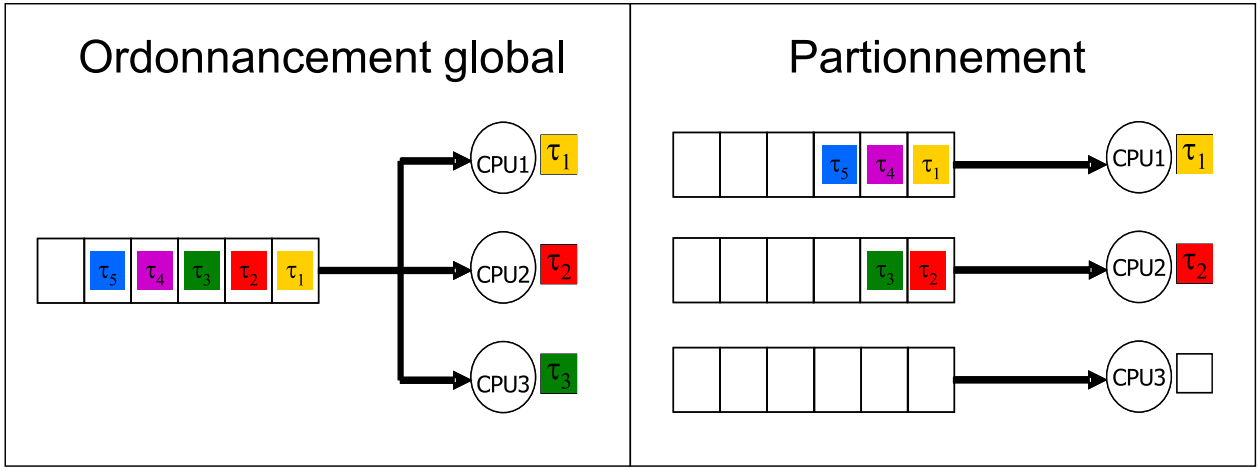


Approches mixtes (6)

- En ligne : parcourir l'arbre de la racine jusqu'aux feuilles :
  1. Un serveur a pour échéances celles des entités le constituant.
  2. Niveau dual : appliquer EDF, les serveurs/tâches primals du serveur dual de plus proche échéance ne doivent pas être exécutés.
  3. Niveau pack : appliquer EDF, le serveur/tâche de plus courte échéance doit être exécuté.
- Exemple : avec 5 processeurs, à l'instant 0, $S1*$, et donc $S2$ et $S3$, puis $T4*$/$T7*$ et donc ne pas ordonnancer $T4/T7$.

Sommaire

Résumé (1)



- Illustration système réparti, multi-coeurs :

1. **Ordonnancement global :** choisir d'abord la tâche, puis placer la tâche sur un des processeurs libres. Approche en-ligne.
2. **Ordonnancement par partitionnement :** placement des tâches pour ordonnancement local, puis vérification éventuelle des délais de bout en bout. Approche hors-ligne.

Résumé (2)

- Ressources différentes : plusieurs processeurs, caches, mémoire/horloge/bus partagées, réseaux, NoC, ...
- Ressources différentes $\Longrightarrow$ nouvelles interférences et donc moins de prédictibilité, moins de précision.

- Nouvelles méthodes d'analyse.
- Ordonnancement global (ex : multi-coeurs) : résultats moins nombreux mais domaine de recherche dynamique.
- Ordonnancement par partitionnement (ex : systèmes répartis) : placement, vérification de contraintes de bout en bout, ordonnancement de messages, ...
- Un peu de pratique ? la majorité des méthodes d'analyses présentées dans ce cours ont été implantées dans Cheddar.

Sommaire

Acronymes

- **MAP**. Manufactoring Automation Protocol.
- **CAN**. Controller Area Network.
- **FIP**. Factory Instrumentation Protocol.
- **FIFO**. First In First Out.
- **MAC**. Medium Acces Control.
- **PROFIBUS**. Process Field Bus.
- **CSMA/CA**. Carrier Sence Multiple Access and Collision Detection Avoidance.
- **TDMA**. Time Division Multiple Access.
- **PAES**. Pareto Archived Evolution Strategy.

Références

[AND 05] James Anderson, Philip Holman, and Anand Srinivasan. « Fair Scheduling of Real-time Tasks on Multiprocessors ». In *Handbook of Scheduling*, 2005.

[BAR 03] S. K. Baruah and S. Funk. « Task assignment in heterogeneous multiprocessor platforms ». Technical Report, University of North Carolina, 2003.

[BAT 97] I. Bate and A. Burns. « Schedulability Analysis of Fixed priority Real-Time Systems with Offsets », 1997.

[BER 07] M. Bertogna. « Real-Time scheduling analysis for multiprocessor platforms ». In *Phd Thesis, Scuola Seprropre Sant Anna, Pisa*, 2007.

[BLA 76] J. Blazewicz. « Scheduling Dependant Tasks with Different Arrival Times to Meet Deadlines ». In. Gelende. H. Beilner (eds), Modeling and Performance Evaluation of Computer Systems, Amsterdam, Noth-Holland, 1976.

[BUR 94] A. Burchard, J. Liebeherr, Y. Oh, and S. H. Son. « Assigning real-time tasks to homogeneous multiprocessor systems ». January 1994.

Références

[CHA 95] S. Chatterjee and J. Strosnider. « Distributed pipeline scheduling : end-to-end Analysis of heterogeneous, multi-resource real-time systems ». 1995.

[CHE 90] H. Chetto, M. Silly, and T. Bouchentouf. « Dynamic Scheduling of Real-time Tasks Under Precedence Constraints ». *Real Time Systems, The International Journal of Time-Critical Computing Systems*, 2(3):181–194, September 1990.

[CIA 99] CIAME. *Réseaux de terrain*. Edition Hermès, 1999.

[COT 00] F. Cottet, J. Delacroix, C. Kaiser, and Z. Mammeri. *Ordonnancement temps réel*. Hermès, 2000.

[COU 94] G. Coulouris, J. Dollimore, and T. Kindberg. *Distributed Systems—Concepts and Design, 2nd Ed.* Addison-Wesley Publishers Ltd., 1994.

[DAV 09] R.I. Davis and A. Burns. « A Survey of Hard Real-Time Scheduling Algorithms and Schedulability Analysis Techniques for Multiprocessor Systems ». In *University of York Technical Report, YCS-2009-443*, 2009.

[DEC 96] T. Decker. « Three Popular Avionics Databuses ». *Real Time Magazine*, (2):29–34, April 1996.

Références

[DHA 78] S. Dhall and C. Liu. « On a real time scheduling problem ». *Operations Research*, 26:127–140, 1978.

[DWY 91] J. Dwyer and A. Ioannou. *Les réseaux locaux industriels MAP et TOP*. Editions Masson, mars 1991.

[GEO 96] L. George, N. Rivierre, and M. Spuri. « Preemptive and Non-Preemptive Real-time Uni-processor Scheduling ». INRIA Technical report number 2966, 1996.

[HEA 96] D. Head. « MIL-STD-1553B ». *Real Time Magazine*, (2):25–28, April 1996.

[KOO 95] P. J. Koopman. « Time Division Multiple Access Without a Bus Master ». Technical Report RR-9500470, United Techologies Research Center, June 1995.

[LEB 95] L. Leboucher and J. B. Stefani. « Admission Control end-to-end Distributed Bindings ». pages 192–208. COST 231, Lectures Notes in Computer Science, Vol 1052, November 1995.

Références

[LEG 03] J. Legrand, F. Singhoff, L. Nana, L. Marcé, F. Dupont, and H. Hafidi. « About Bounds of Buffers Shared by Periodic Tasks : the IRMA project ». In the 15th Euromicro International Conference of Real Time Systems (WIP Session), Porto, July 2003.

[LIU 73] C. L. Liu and J. W. Layland. « Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environnment ». *Journal of the Association for Computing Machinery*, 20(1):46–61, January 1973.

[OH 93] Y. Oh and S. H. Son. « Tight Performance Bounds of Heuristics for a Real-Time Scheduling P roblem ». Technical Report, May 1993.

[OH 95] Y. Oh and S.H. Son. « Fixed-priority scheduling of periodic tasks on multiprocessor systems ». 1995.

[PAL 03] C. Palencia and M. Gonzalez Harbour. « Offset-Based Response Time Analysis of Distributed Systems Scheduled under EDF ». In *15th Euromicro Conference on Real-Time Systems (ECRTS'03)*, 2003.

[PUJ 95] G. Pujolle. *Les réseaux* . Editions Eyrolles, décembre 1995.

Références

[RIC 01] P. Richard, F. Cottet, and M. Richard. « On line Scheduling of Real Time Distributed Computers With Complex Communication Constraints ». 7th Int. Conf. on Engineering of Complex Computer Systems, Skovde (Sweden), June 2001.

[STR 96] H. Strass. « Factory Floor Networks PROFIBUS : the natural choice ». *Real Time Magazine*, (2):6–8, April 1996.

[TIN 92] K. Tindell. « Using Offset Information to Analyse Static Priority Pre-Emptively Scheduled Task Sets ». In *YCS. 182, Dept of Computer Science, University of York*, 1992.

[TIN 94] K. W. Tindell and J. Clark. « Holistic schedulability analysis for distributed hard real-time systems ». *Microprocessing and Microprogramming*, 40(2-3):117–134, April 1994.

Références

[UPE 94] P. Upendar and P. J. Koopman. « Communication Protocols for Embedded Systems ». *Embedded Systems Programming*, 7(11):46–58, November 1994.

[XU 90] J. Xu and D. Parnas. « Scheduling Processes with Release Times, Deadlines, Precedence, and Exclusion Relations ». *IEEE Transactions on Software Engineering*, 16(3):360–369, March 1990.

[ZEL 96] H. Zeltwanger. « CAN in industrial Applications ». *Real Time Magazine*, (2):20–24, April 1996.

Remerciements

- Certains de ces transparents sont extraits ou inspirés des cours de C. Pagetti et de J. Goossens. Merci à ces auteurs !