Conception et Validation des Systèmes Temps-Réel : Architecture, Parallélisme et Sûreté de Fonctionnement

V. David
Expert Senior IRSN
Expert Systèmes Temps-Réel, Sûreté de Fonctionnement
Précédemment Krono-Safe, Directeur Technique
Fondateur du LaSTRE du CEA LIST
Pr. INSTN

Introduction à l'Ordonnancement

Programmation multitâche

Un Système Temps-Réel dispose :
- de plusieurs ressources
- capables d'assurer 1 fonction à la fois

afin de remplir :
- plusieurs fonctionnalités

1 objectif fcnl = 1 ou plusieurs tâches
- 1 tâche = 1 ou plusieurs fonctions

La mise en oeuvre du système
suppose une allocation des tâches
sur les diverses ressources
au cours du temps

Faire cette allocation au cours du temps,
c'est *ordonnancer* les traitements

Gérer un objectif temps-réel global

- des échelles de temps différentes
(les traitements courts passent avant les longs)
- des degrés de criticité différents
(les traitements critiques passent en premier)

Objectifs souvent contradictoires

- maximiser le nombre de fonctionnalités correctement acquittées
- minimiser le coût (dimensionnement)

Classification des tâches par importance

Critiques :
- doivent toujours être assurées
(garantir des propriétés de sûreté)

Essentielles :
- doivent être assurer autant que possible,
c'est à dire au moins de temps en temps
(garantir des propriétés de vivacité)

Dans tous les cas, pour un système Temps-Réel,
il faut assurer la ponctualité de tous les traitements

Optionnelles : ...

100% des tâches critiques doivent respecter leurs contraintes (preuve)

Pour les tâches essentielles
"best-effort"

Les ressources peuvent être :
- réquisitionnables
- non réquisitionnables
- multiples ou non

=> complexité du problème d'ordonnancement (NP-complet dans le cas général)

	Tortue	Lièvre
Vitesse	1	10
Durée Commutation	1	0
Priorité	A l'Echéance	Au Premier

Tâche A : Durée=270/1 Echéance=t+320

Tâche B : Durée=15/1 Echéance=t+21

A arrive juste avant B...

Attributs temporels d'une tâche :

- date de début au plus tôt (demande)
- date de fin au plus tard (échéance)
- durée d'exécution (besoin)

Alors, on peut déduire :

- date de début au plus tard
- durée jusqu'à la date de début au plus tard
- date de fin au plus tôt (pb de réquisition)

Relations entre les tâches :

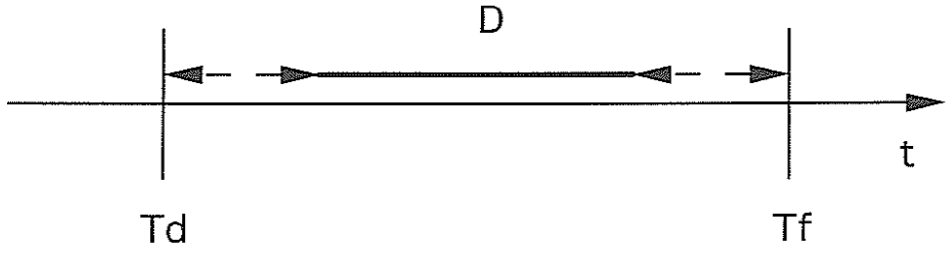
- graphe de dépendance

(calcul des dates au plus tôt et au plus tard)

- concurrence explicite
- concurrence implicite
- atomicité, sérialisation...

Marge statique :

Ms = (Tf - Td) - D

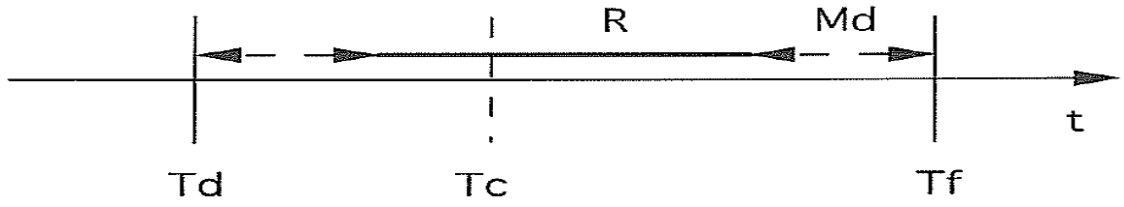


- Ms > 0

- Si Ms = 0, pas de choix

Marge dynamique :

Md = (Tf - Tc) - R



- Md = Laxité
- Md évolue dynamiquement
  - pour la tâche active
  - pour les tâches non actives

<u>Plan d'ordonnancement</u> :
c'est une méthode de prévision pour l'allocation des ressources (conception)

Le plan est dit *optimal* si toutes les contraintes temporelles sont respectées

On dit qu'il y a *surcharge* , lorsque le volume des tâches à ordonnancer est tel que tous les plans d'ordonnancement conduisent au non respect d'au moins une tâche (il n'existe pas de plan optimal)

Un *algorithme* d'ordonnancement est dit optimal, si pour une classe de problème donnée, il produit des plans optimaux

Ordonnancement statique : la priorité d'une tâche ne change pas pendant l'exécution
-> décision hors-ligne

Ordonnancement dynamique : la priorité d'une tâche varie dynamiquement
-> décision à l'exécution

→ heuristique simple du 1 er ordre : on associe, à chaque événement susceptible de modifier le plan, une priorité à chaque tâche en mesure de s'exécuter

→ heuristique d'ordre supérieure : idem, mais on évalue en ligne les conséquences de plusieurs choix jusqu'à un futur donné

Statique

Avantages :

+ l'horizon de la prévision peut être infini

+ la connaissance du plan est totale

(garantie, flot d'instructions unique)

=> vérification / simulation

(ponctualité, relations entre tâches)

Inconvénients :

- rigide, nécessite une régularité

des traitements, peut être inadaptée

- sous optimale, peut être inefficace

Principalement utilisé en temps-réel et sûreté (démonstration)

Dynamiques

+ flexibles, optimaux car informé

- prédictibles ?, instabilités, non oisifs

Complexes, et donc, a priori moins sûrs...

Un énoncé de problème

[périodique, sporadique]

[avec dépendance ou sans]

[etc.]

=

Un choix parmi les hypothèses

Diverses classes de problème
avec des solutions connues,
souvent optimales

(selon les hypothèses retenues)

Principaux algorithmes connus

Statique :

- **RMS**

Dynamique :

- **FCFS**
- **RR**
- **HPF**
- **LLF**
- **EDF**

1 Introduction à l'Ordonnancement

- Les Objectifs

2. Ordonnancement statique

- RMS

3. Ordonnancement dynamique

- RR / FCFS
- HPF
- LLF / EDF

4. Autres cas

- Dépendances
- Multiprocesseur

Rate Monotonic Scheduling

Liu&Layland (73)

Hypothèse de modèle (Hl)

- statique
- chaque tâche est périodique
- pas de dépendance entre tâche
- l'échéance est la période
- la priorité est l'inverse de la période
- réquisition

Hypothèse de faisabilité (H2)

Il existe une condition suffisante (CS)

→ordonnancement sûr si critère vérifié

NB : durées d'exécution connues (majorant ou WCET)

Critère théorique

- n tâches
- Ci=durée
- Ti=période

Analyse du taux d'occupation (W) CPU :
si W inférieur a U(n), RMS est optimal
(H1 et H2)

U(n)=n*(2^(1/n)-1)
U(1)=1
U(2)=0.83
U(3)-0.78
U(∞)=0.69

Condition suffisante

$W = \sum Ci/Ti$

U(n)=n*(2^(1 /n)-1)

CS : $W \leq U(n)$

et trivialement, on a une CN : $W \leq 1$

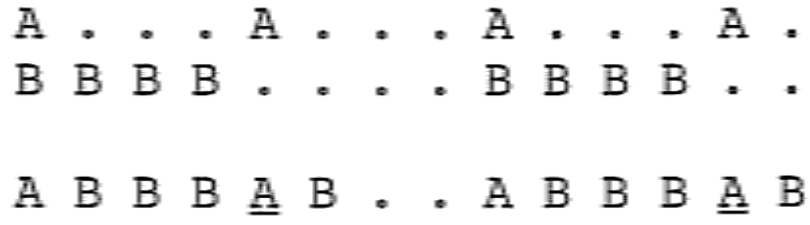
Pour W entre CS et CN :
Il faut réaliser le plan d'ordonnancement
sur un horizon fini (le PPCM des périodes)

A : C=1 T=4 —> P=2

B : C=4 T=8 —> P=1

W = 1/4+4/8 = 0.75

CS : W ≤ 0.83



Conclusion RMS

+ statique (prédéterminé : sûr, fiable)
+ peut être étendu à une tâche apériodi
(Serveur Sporadique)

- lois très simples
- pas d'asynchronisme
- aspects multitâches très réduits

Très proche de la programmation en bc
mais conception plus aisée (simple et ef
(toutefois PPCM = Cycle)

1 Introduction à l'Ordonnancement

- Les Objectifs

2. Ordonnancement statique

- RMS

3. Ordonnancement dynamique

- RR / FCFS
- HPF
- LLF / EDF

4. Autres cas

- Dépendances
- Multiprocesseur

Round-Robin

On partage le temps de façon "équitable«

Principe du tourniquet sur les tâches en mesure de s'exécuter

Au bout de k unité de temps sur la même tâche

+ vivacité
- Ponctualité

Ex : OCCAM / TRANSPUTERS
En pratique, RR est couplé avec HPF

First Corne / First Served

Très simple et très rudimentaire

+ vivacité
-- ponctualité

Architecture client/serveur

Utile pour garder un ordre de traitement implicite (pour les E/S)

Ex : spooler d'impression...

High Priority First

Les tâches ont une priorité statique

La tâche de plus haute priorité en mesure de s'exécuter prend la main

++ très proche H/W (E/S asynchrones)
+ ponctualité pour une tâche
- pour les autres ?
- sûreté
- vivacité

Changement possible de priorité en ligne
(plus pour descendre = vivacité)
=> amélioration des défauts (un peu)

Tâche A : D=2 P=3
- prête à s'exécuter
- mais requête à !a tâche C pendant ses traitements (dépendance)

Tâche B : D=2 P=2
- prête à s'exécuter

Tâche C : D=1 P=1
- en attente

Least Laxity First

Hypothèse de modèle (H1)

Hypothèse de faisabilité (H2)

• dynamique

- tâche apériodique

optimal si pas de surcharge

- pas de dépendance entre tâche
- échéances connues
- durées connues
- la priorité est l'inverse de la laxité
- préemption

Earliest DeadLine First

Liu&Layland (73)

Hypothèse de modèle (H1)

• dynamique
- tâche apériodique
- pas de dépendance entre tâche
- échéances connues
- *durées inconnues*
- la priorité est l'inverse de la laxité
- préemption

Hypothèse de faisabilité (H2)

optimal si pas de surcharge

1 Introduction à l'Ordonnancement

Les Objectifs

2. Ordonnancement statique

RMS

3. Ordonnancement dynamique

RR / FCFS

HPF

LLF / EDF

4. Autres cas

Dépendances

Multiprocesseur

Hypothèses (presque) jamais vérifiées :
- l'indépendance des traitements
- dérive des lois événementielles
=> stabilité / prédictibilité(1 ou2)...
..."surcharge ?«

Algorithmes étudiés
statique :
- RIMS
dynamique :
- FCFS
- RR
- HPF
- LLF
- EDF

Cas multiprocesseurs : système distribué
- les temps de latences ne sont plus négligeables (communications, routages, migrations)
- l'état global n'est pas observable (connaissance partielle)
=> obsolescence des décisions dynamiques (équilibrage de charge)

Cas multiprocesseurs à bus partagé
- hypothèses simplificatrices
- classe des « processeurs multicœurs »

Hypothèses (presque) jamais vérifiées :
- l'indépendance des traitements
- dérive des lois événementielles
=> stabilité / prédictibilité(1 ou2)...
..."surcharge ?«

Algorithmes étudiés
statique :
- RIMS
dynamique :
- FCFS
- RR
- HPF
- LLF
- EDF

Cas multiprocesseurs : système distribué
- les temps de latences ne sont plus négligeables (communications, routages, migrations)
- l'état global n'est pas observable (connaissance partielle)
=> obsolescence des décisions dynamiques (équilibrage de charge)

Cas multiprocesseurs à bus partagé
- hypothèses simplificatrices
- classe des « processeurs multicœurs »

Cas mono CPU :

l'approche intuitive "best-effort"
fondée sur les marges stat/dyn
conduit souvent au résultat optimal

⇒ EDF(LLF)
⇒ Avec partitionnement temporel

Cas multi CPU :

- intuition généralement fausse
- e.g. Global-EDF n'est pas optimal…

l'ordonnancement demeure un problème ouvert
(mais il existe des outils pour trouver
des solutions approchées)

=> intuition entropique
- diminuer les contraintes
- sans en rajouter

"faire les choix forcés, retarder les autres"