Programmation Temps Réel

Gestion des situations de surcharge

Yann Thoma

Reconfigurable and Embedded Digital Systems Institute
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

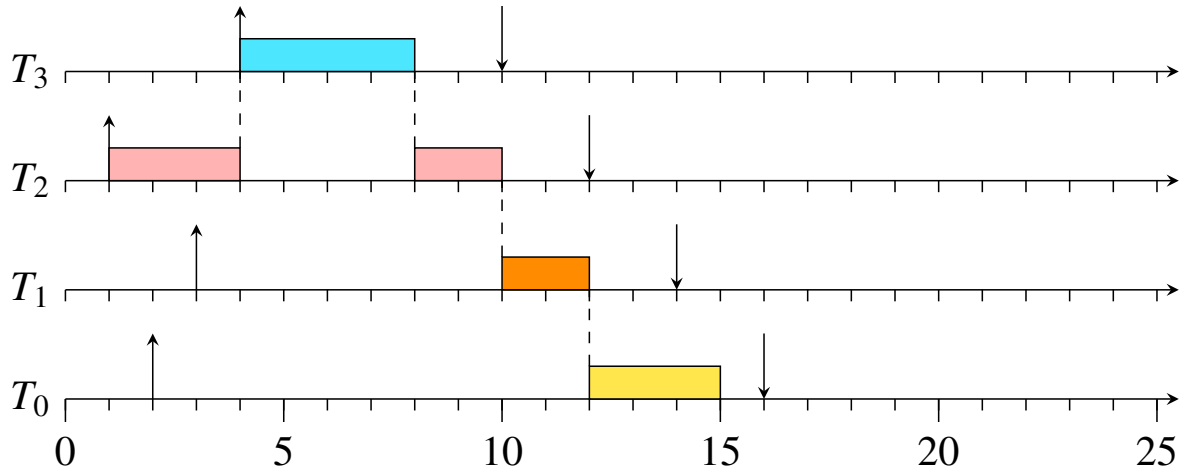
Septembre 2017

Types de surcharge

- Surcharge temporaire due à un surplus d'activation
  - Dans un système basé sur des événements
  - Un événement déclenche une tâche qui met en péril le schéma d'ordonnancement
  - EDF y est très sensible (Effet domino)
- Surcharge temporaire due à un temps d'exécution trop long
  - Une tâche périodique ou apériodique prend plus de temps que prévu
  - RM: seules les tâches de priorité inférieure sont affectées
  - EDF: n'importe quelle tâche peut être affectée
- Surcharge permanente dans le cas de tâches périodiques
  - Le facteur d'utilisation est plus grand que 1
    - Mauvais design
    - Arrivée d'une nouvelle tâche périodique
    - Modification de la période d'une tâche

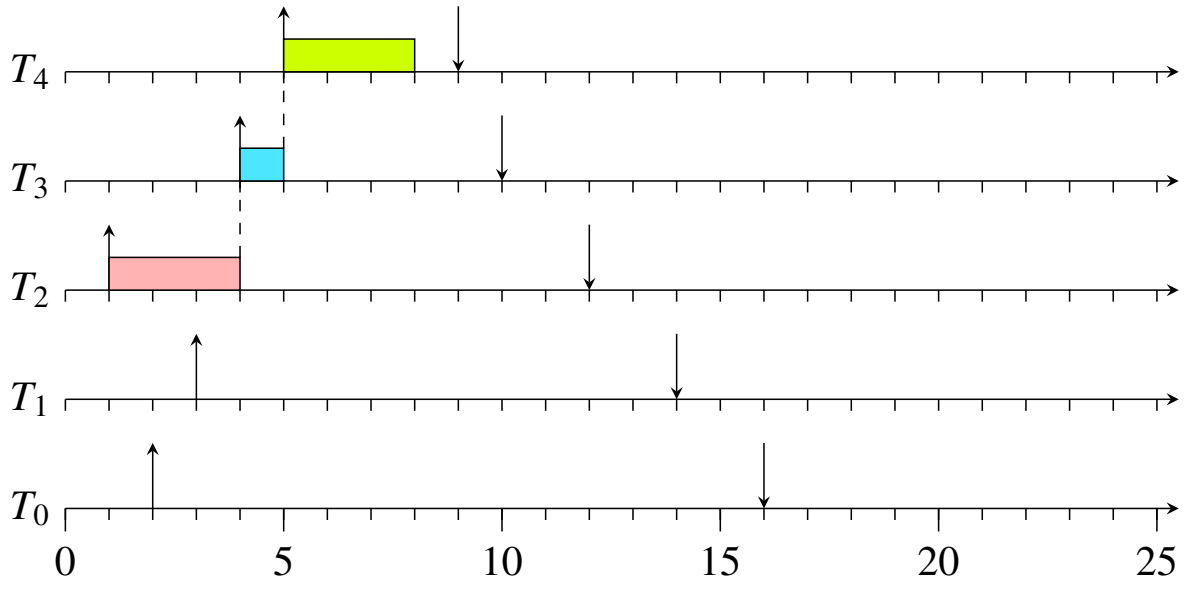
Surcharge temporaire due à l'activation (1)

- Politique EDF
- $C_0 = 3, C_1 = 2, C_2 = 5, C_3 = 4$



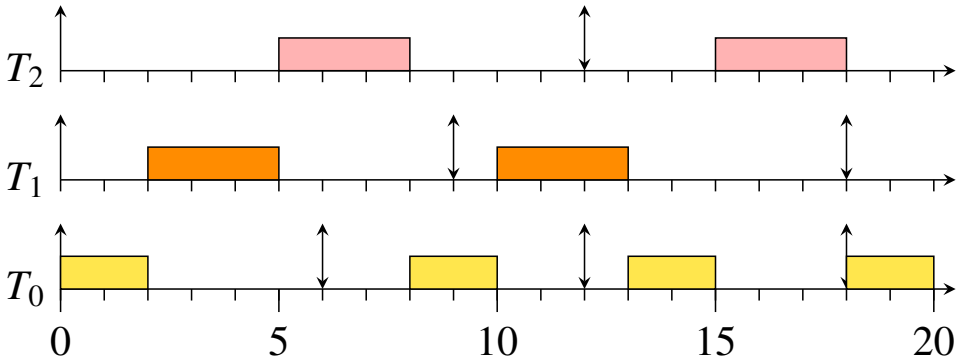
Surcharge temporaire due à l'activation (2)

- Politique EDF
- Activation d'une nouvelle tâche, T_4: $C_4 = 3$



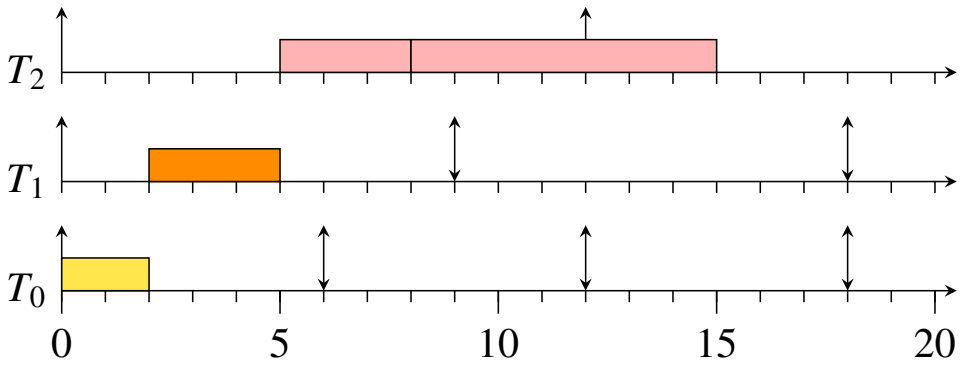
Surcharge temporaire due à une exécution trop longue (1)

- Politique EDF



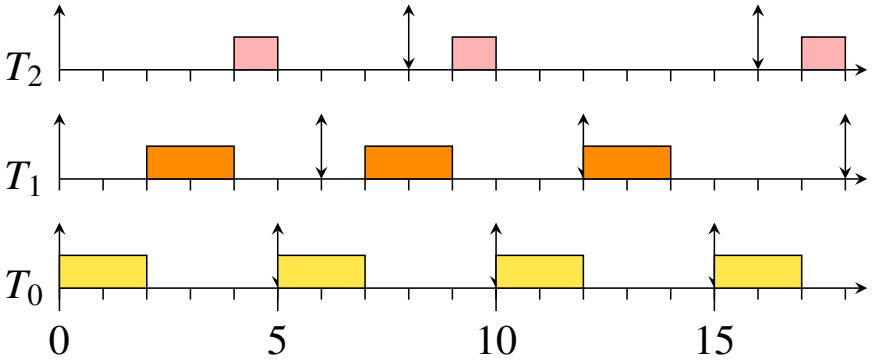
Surcharge temporaire due à une exécution trop longue (2)

- Politique EDF
- Surcharge de la tâche T_2: raisons?



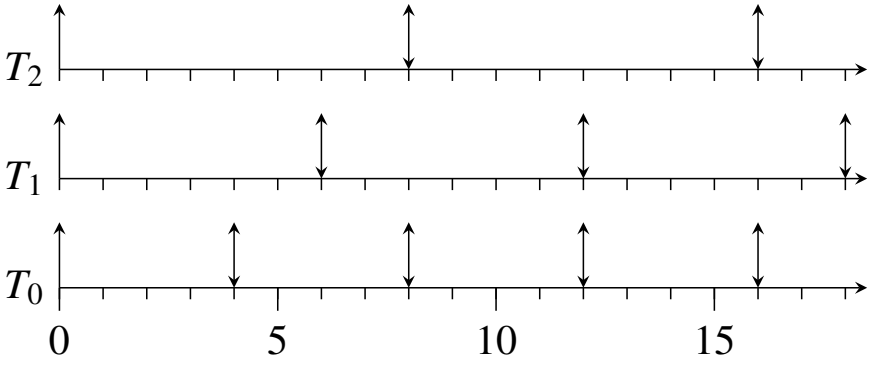
Surcharge permanente (1)

- Politique RM
- Ordonnançabilité? (Hyperbolic Bound)



Surcharge permanente (2)

- Politique RM
- La période de la tâche T_0 passe à 4. Ordonnançabilité? (Hyperbolic Bound)



Introduction

- EDF et RM ne gèrent pas la surcharge
- Que faire dans une telle situation?
  - Détecter la surcharge
  - Proposer une réorganisation

Détection de la surcharge (1)

- Par calcul de la laxité
  - Soit l'ensemble des tâches $\Gamma = T_i(t, C_i(t), d_i)$ actives au temps t
  - Evalué au temps t de lancement d'une tâche
  - les tâches T_i triées par ordre croissant d'échéance ($i < j \Leftrightarrow d_i < d_j$)

$$LC_i(t) = d_i - \sum_j C_j(t), d_j \leq d_i \quad (1)$$

$$LP(t) = \min_i(LC_i(t)) \quad (2)$$

  - Si la laxité du système $LP(t) < 0$, alors une tâche au moins ne respecte pas son échéance
  - Il s'agit des tâches avec $LC_i(t) < 0$

Détection de la surcharge (2)

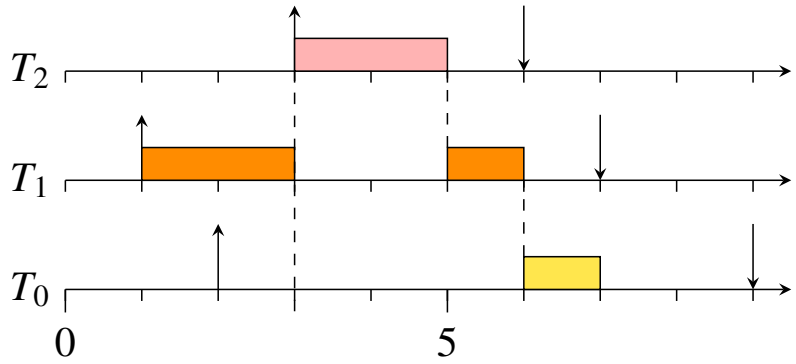
- Par calcul de la charge
  - Soit l'ensemble des tâches $\Gamma = T_i(t, C_i(t), d_i)$ actives au temps t
  - La charge processeur est définie par

$$\rho_i(t) = \frac{\sum_{d_k \leq d_i} c_k(t)}{(d_i - t)} \quad (3)$$

$$\rho(t) = \max_i(\rho_i(t)) \quad (4)$$

  - Si $\rho(t) > 1$, alors une tâche au moins ne respecte pas son échéance
  - Il s'agit des tâches avec $\rho_i(t) > 1$

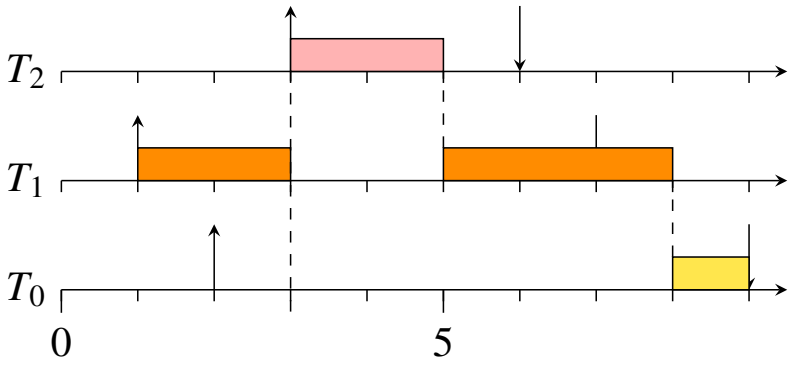
Détection de surcharge: exemple (1)



Evaluation au temps 3

$$\rho_2(3) =$$
$$\rho_1(3) =$$
$$\rho_0(3) =$$
$$\rho(3) =$$

Détection de surcharge: exemple (2)



$$\begin{aligned} \rho_2(3) &= \\ \rho_1(3) &= \\ \rho_0(3) &= \\ \rho(3) &= \end{aligned}$$

Types d'algorithmes

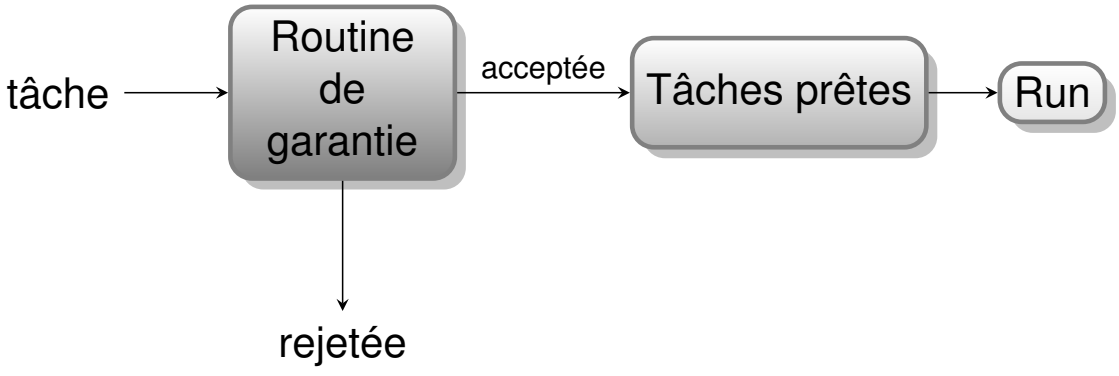
- Les algorithmes de gestion de la surcharge peuvent être décomposés en trois classes:
  - Meilleur effort (Best effort)
  - Routine de garantie (guaranteed)
  - Ordonnancement robuste (robust scheduling)

Classe d'algorithme: Best effort

- Lors de l'arrivée d'une tâche, elle est placée dans la liste des tâches
- Seules les priorités des tâches peuvent permettre de garantir des échéances sur certaines tâches

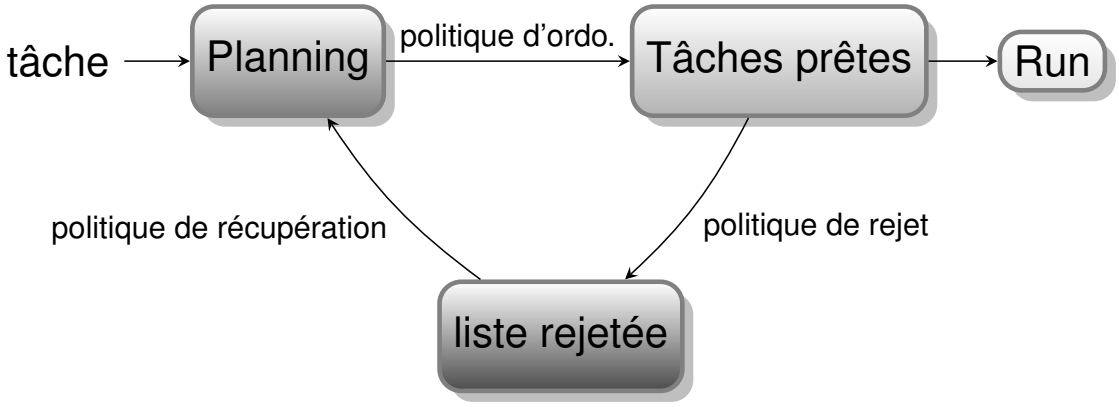
Classe d'algorithme: Guaranteed

- Lors de l'arrivée d'une tâche, un test d'acceptation est exécuté
- S'il passe, la tâche est ajoutée à la liste
- Sinon la tâche est rejetée



Classe d'algorithme: Robust scheduling

- Lors de l'arrivée d'une tâche, un test d'acceptation est exécuté
- S'il passe, la tâche est ajoutée à la liste
- Sinon l'*importance* des tâches est utilisée pour supprimer certaines tâches



Solutions pour tâches périodiques: Mécanisme à échéance

- Une tâche est décomposée en:
  - Une version primaire
    - à temps indéterminé
    - bonne qualité de service
  - Une version secondaire
    - à temps déterminé
    - résultat acceptable
- 2 politiques:
  - Première chance
    - Priorité des tâches secondaires $>$ Priorité des tâches primaires
    - Primaires ordonnancées dans le temps restant
  - Deuxième chance
    - Les primaires sont exécutées avant les secondaires
    - Si la primaire est exécutée, la secondaire ne l'est pas
    - Il faut garantir que la secondaire puisse s'exécuter si nécessaire
- Pour la qualité, il faut quand-même assurer quelques primaires

Solutions pour tâches périodiques: méthode du calcul approché

- Une tâche est décomposée en:
  - Une partie *Obligatoire*
    - Fournit un résultat approché
    - Doit s'exécuter avant l'échéance
  - Une partie *Optionnelle*
    - Affine le résultat
    - Exécutée s'il y a du temps à disposition
- Avantages de cette approche:
  - Moins de coordination entre tâches
- Pour la qualité, il faut quand-même assurer un certain taux de parties optionnelles

Solutions pour tâches quelconques: Types de solutions

- Pour Guaranteed et Robust scheduling
  - Le test d'acceptation permet d'éviter l'effet domino
- Premier courant:
  - Une importance est donnée aux tâches
- Deuxième courant:
  - L'importance des tâches varie en fonction du temps

1ère approche

- Type routine de garantie
- Lors de l'arrivée d'une nouvelle tâche
  - La surcharge (ou laxité) est évaluée
  - S'il y a surcharge, la tâche n'est pas exécutée
- Problème:
  - Certaines tâches peuvent ne jamais être exécutées
  - Certaines tâches critiques doivent forcément s'exécuter

2ème approche

- Type ordonnancement robuste
- Une tâche possède une *importance*
- Lors de l'arrivée d'une nouvelle tâche
  1. La surcharge (ou laxité) est évaluée
  2. Si nécessaire, la tâche la moins importante est supprimée
  3. Retour au point 1
- Problème:
  - Un tâche non exécutée peut mettre en péril le système
  - Comment supprimer une tâche en cours?

3ème Approche

- On reprend l'approche précédente
- Les tâches sont décomposées en 4 modes d'exécution
  - Normal
    - Exécution standard
  - Révocation
    - Exécuté lorsque la tâche est supprimée durant son fonctionnement
  - Ajournement
    - Exécuté lorsque la tâche est supprimée avant son exécution
  - Faute temporelle
    - Exécuté lorsqu'une faute temporelle est détectée

3ème Approche

Exemple

```
Tâche T1:
  Mode normal: (C=10, ajournable, révocable, Imp=5)
    Acquérir(Capteur);
    Valeur=Lire(Capteur);
    Restituer(Capteur);
    Temp=Calcul(Valeur);
    Send(Temp,T2);
  Mode ajournement: (C=2, obligatoire, Imp=5)
    Temp=old_temp*facteur_de_correction;
    Send(Temp,T2);
  Mode révocation: (C=3, obligatoire, Imp=5)
    Restituer(Capteur);
    Ajourner(T2);
Fin tâche T1;
```

2ème courant: Algorithme RED (exemple)

- Une tâche est caractérisée par $T_i(C_i, D_i, M_i, V_i)$
  - C_i: Coût
  - D_i: Echéance
  - M_i: Tolérance d'échéance, temps de dépassement acceptable
  - V_i: Importance
- Lorsqu'une tâche est introduite:
  - Les laxités sont calculées
  - Le temps excédent E_i est calculé
    - $E_i = max(0, -(L_i + M_i))$
  - $E_{max} = \max_i(E_i)$
  - Si $L_i < 0$ pour toutes les tâches, OK
  - Si $E_{max} = 0$, OK
  - Sinon, sélectionner les tâches à supprimer
  - Les tâches supprimées sont placées dans une liste
- Quand une tâche termine, on évalue la possibilité d'exécuter une tâche de la liste

Modèle élastique

- Dans le cas de tâches périodiques
- Une élasticité est associée à chaque tâche
- La fréquence des tâches en légèrement modulable, en fonction de l'élasticité