Tolérance aux fautes temporelles et amélioration du comportement d'un système Java temps réel

JDIR 2005

Damien Masson

Institut Gaspard-Monge, Université de Marne-La-Vallée

13-15 Decembre

Java ?

- Volonté d'apporter les avantages de Java au monde du temps réel :
  - Syntaxe claire
  - Niveau d'abstraction élevé
  - Plate-forme indépendant
- Mais Java n'est pas directement utilisable :
  - Peu de contrôle du programmeur sur la mémoire
  - Ramasse miette non prévisible
  - Pas d'assurance sur la politique d'ordonnancement

Real-Time Specification for Java (RTSJ)

- 2001 - 2005
- Compatibilité ascendante
- Pas d'extension de la syntaxe
- WORA devient WOCRAM

Concrètement

- Ce n'est qu'une spécification : pas d'implantations complète de la norme
- Plusieurs environnements à notre disposition :
  - RI (Reference Implementation) : machine virtuelle développée par timesys, gratuite mais pas open-source
  - jRate : extension du GNU GCJ compiler, GPL
  - jamaïcaVM, Flex, OVM, Mackinac project ...
- Tous reposent sur une architecture temps réel sous jacente

Condition de charge

- Hypothèses : Système de tâches indépendantes, périodiques, ordonnanceur à priorités fixes préemptif
- Notations : une tâche τ_i possède un coût C_i, une période T_i, une échéance D_i et une priorité P_i
- La charge U du système s'exprime par :

$$U = \sum_{i=1}^{N} \frac{C_i}{T_i} \quad (1)$$

- Condition de charge :
  - $U > 1 \Rightarrow$ le système n'est pas faisable
  - $U \leq 1 \Rightarrow$ condition non suffisante

Détermination du pire temps de réponse

Le temps de réponse pire cas d'une tâche, noté $WCRT_i$, est le maximum des temps de réponses des instances de la tâche.
On se place dans le cadre d'une activation synchrone.

- Si $\forall i, D_i \leq T_i$ le pire temps de réponse est atteint dès la première instance
- Sinon, pour chaque tâche, il faut calculer les temps de réponse de toutes les instances jusqu'à en trouver une qui n'influence pas la suivante

$$R_{(i,l)}^{n+1} = C_i + \sum_{j \ tq \ \tau_j \in HP(\tau_i)} \left\lceil \frac{R_{(i,l)}^{n}}{T_j} \right\rceil * C_j \qquad (2)$$

Le système est faisable si : $\forall i, WCRT_i \leq D_i$

Implantation

- Specification : `RealtimeThread.addToFeasibility()`, `Scheduler.isFeasible()`
  - RI : facile de trouver des systèmes non faisable pour lesquels répond faisable
  - jRate : méthodes non implantées
- `fr.umlv.masson.realtime.extended.*`
- Sources bientôt disponibles sur `http://igm.univ-mlv.fr/~masson/RTSJ`

Pourquoi des fautes ?

- Faute : une tâche ne respecte pas l'un de ses paramètres - souvent son coût
- Défaillance : en temps réel dur, c'est le dépassement d'une échéance

Nous avons implanté un contrôle d'admission, si une tâche est admise, c'est que le système est faisable !

- Comment le coût (C_i) des tâches est-il obtenu ? une mauvaise estimation a pu être faite
- Facteurs extérieurs au système ...

Que dit la spécification ?

- Lors de la création d'un `RealtimeThread`, on fournit un `ReleaseParameter` qui contient en plus des paramètres de la tâche les champs :
  - `deadlineMissHandler`
  - `costOverrunHandler`
- En pratique :
  - le dépassement de coût n'est pas détecté
  - le handler de dépassement d'échéance ne préempte pas la tâche

Comment détecter un dépassement de coût ?

- Nécessite de savoir à tout instant combien de ressource a consommé chaque tâche $\Rightarrow$ modification de l'ordonnanceur
- Dépassement du $WCRT_i \Rightarrow$ dépassement de C_i
- Les pire temps de réponses sont des dates relatives aux activations, et non des durées
- Implantation par des `PeriodicTimer` lancés en même temps que les tâches, même période, offset égal au WCRT
- Surcharge de :
  - `RealtimeThread.waitForNextPeriod()`
  - `RealtimeThread.start()`

Trois catégories de fautes

1. Fautes ne provoquant aucune défaillance
2. Fautes provoquant la défaillance de la tâche fautive uniquement
3. Fautes provoquant une série de défaillances en cascade au sein des tâches moins prioritaires

Nous souhaitons laisser la tâche fautive s'exécuter tant que la faute n'entre pas dans la troisième catégorie.

Nous appellerons tolérance, A_i, la durée de dépassement du pire temps de réponse autorisée pour la tâche τ_i.

Tolérances

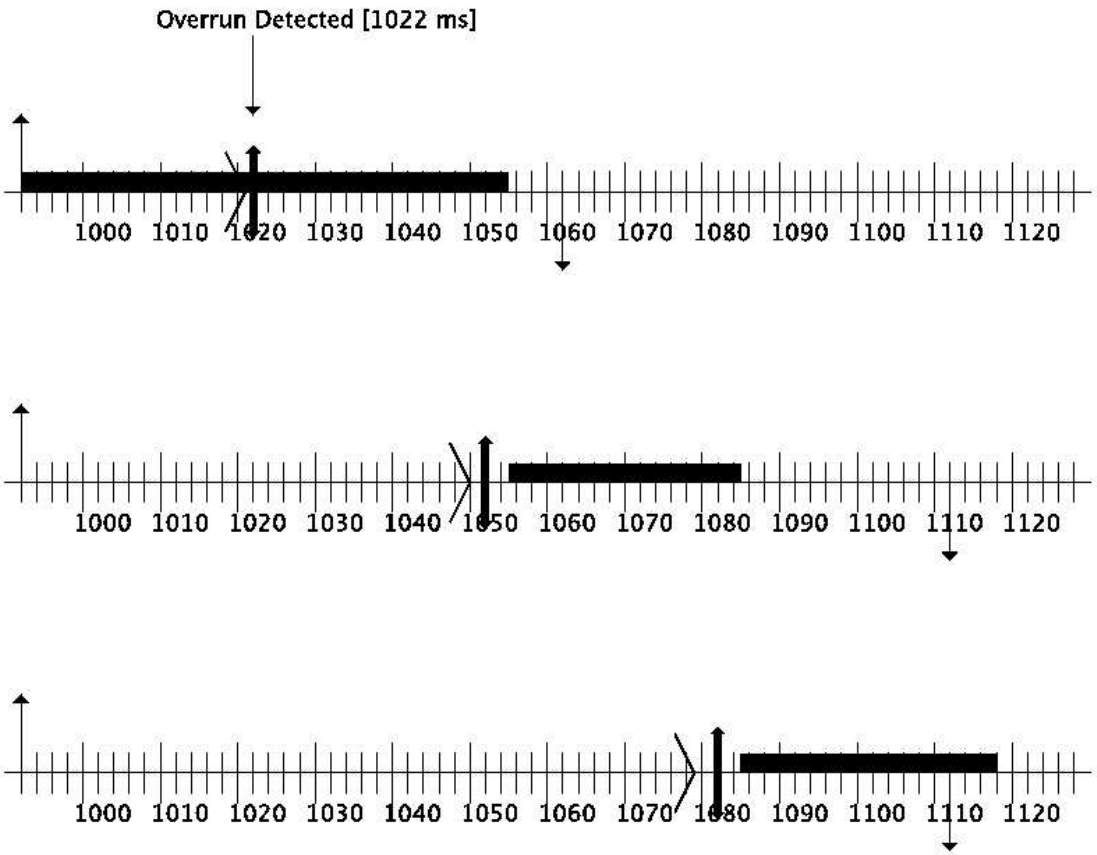
Deux approches pour calculer, pendant le contrôle d'admission, un facteur de tolérance :

- Calcul du coût maximum que l'on peut ajouter à toutes les tâches
- Calcul du coût maximum que l'on peut ajouter à une seule tâches

Dans les deux cas, on effectue une recherche dichotomique validée par l'algorithme de faisabilité.

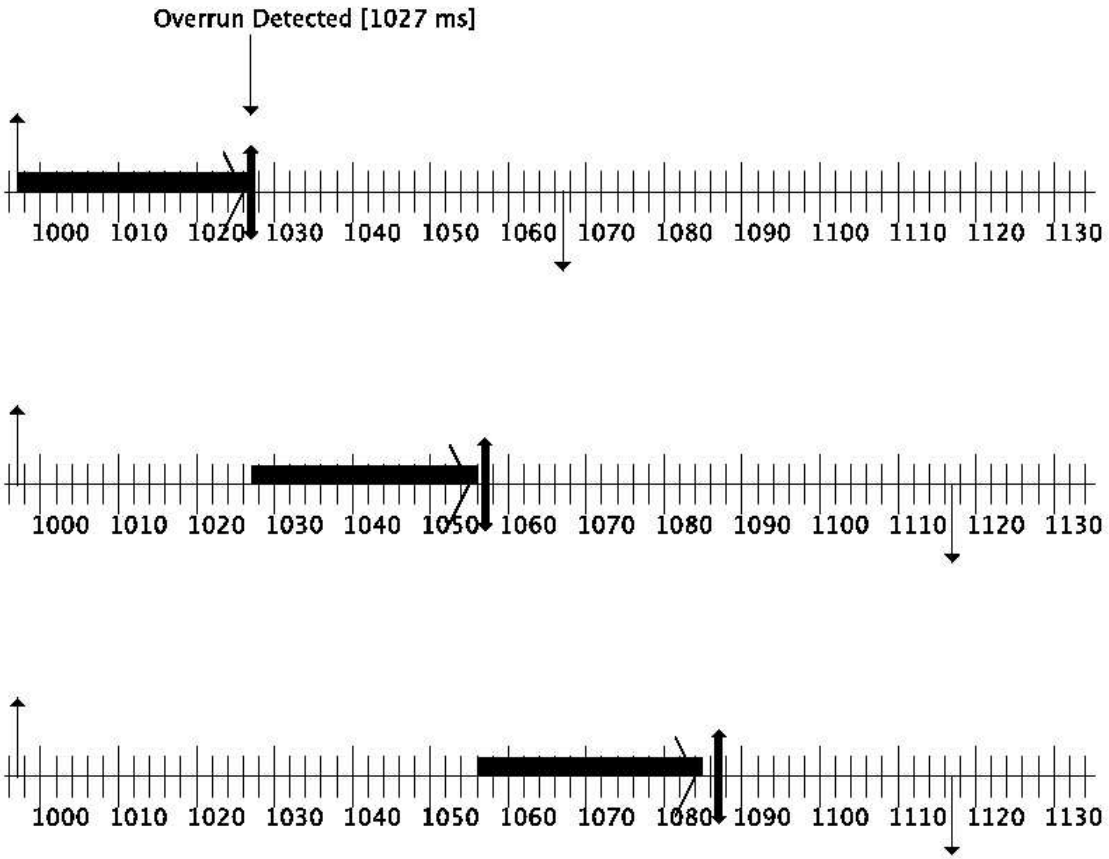
Résultats I

Detection sans traitement :



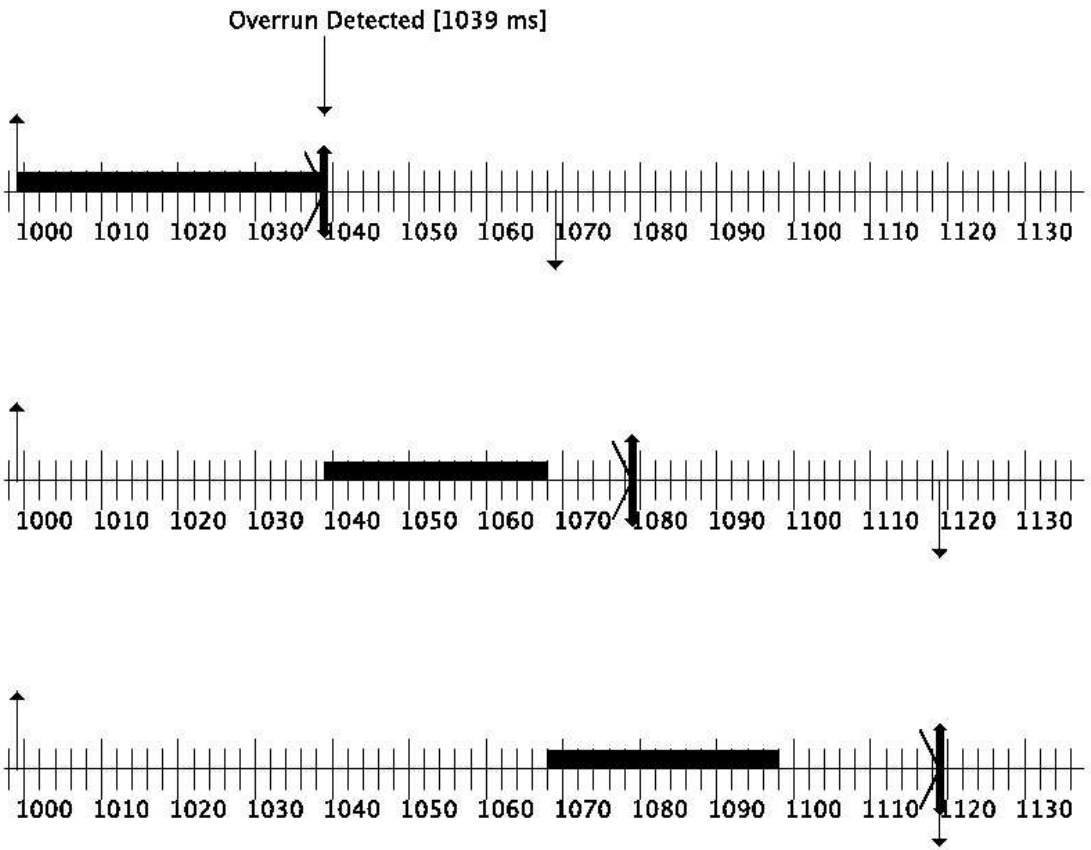
Résultats II

Arrêt immédiat :



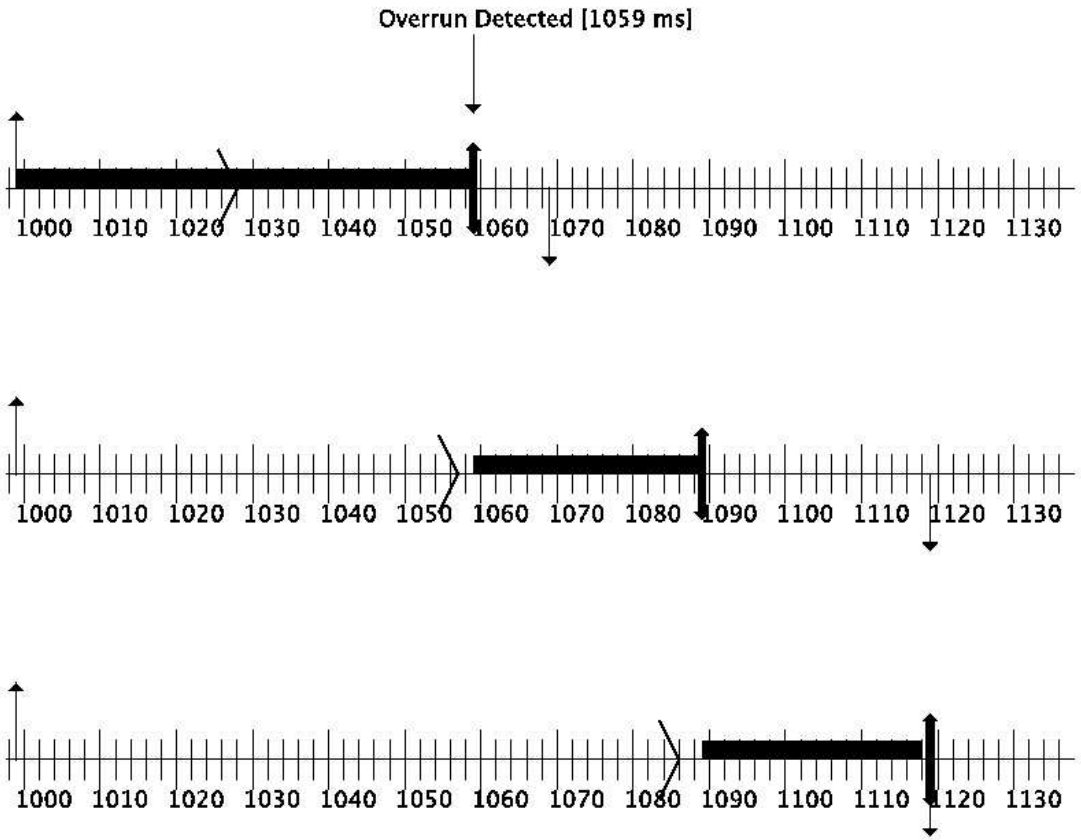
Résultats III

Tolérance accordée à chaque tâche :



Résultats IV

Tolérance accordée à une tâche :



Bilan

- Etat de l'art sur l'analyse de faisabilité et implantation correcte des méthodes spécifiées par la norme, vérifiées avec jRate
- Réutilisation du calcul des $WCRT_i$ pour définir et détecter un type de faute
- Définition et calcul d'un facteur de tolérance
- Exécution sur un exemple : maximisation du temps d'exécution avant qu'une faute ne provoque une défaillance

A suivre

- Influence de la tolérance sur des systèmes avec
  - Contraintes de précédence
  - Partage de ressource (fautes en section critique ?)
- Surestimation de coût : réaffectation des ressources ?
- Systèmes dynamiques : plus de contrôle d'admission mais une détection de surcharge
- Autre type de faute : période minimale d'une tâche sporadique non respectée