Ecole Doctorale STIM
Sciences et Technologies de l'Information et des Matériaux.

Spécialité : Systèmes d'aide à la décision.
Laboratoire : LINA.
Equipe : Ordonnancement temps-réel

ORDONNANCEMENT TEMPS-REEL DYNAMIQUE AVEC CONTRAINTES DE QUALITE DE SERVICE

Marchand, Audrey
Mél : audrey.marchand@univ-nantes.fr

Résumé : Cet article s'intéresse à la simulation et à l'évaluation d'algorithmes d'ordonnancement temps-réel dynamiques sous des contraintes de Qualité de Service (QoS). Les travaux décrits concernent plus précisément l'ordonnancement conjoint de tâches périodiques à pertes contraintes et de tâches apériodiques (tâches non périodiques survenant à un instant non prédéterminé dans le temps). L'approche présentée vise à exploiter les pertes autorisées au niveau des tâches périodiques dans le but de minimiser le temps de réponse moyen des requêtes apériodiques, par le biais de l'utilisation du serveur de tâches apériodiques Earliest Deadline as Late as possible (EDL).

Mots clés : *Systèmes temps-réel, ordonnancement, qualité de service, Earliest-deadline, tâches périodiques, serveur de tâches apériodiques.*
Collaborations : Ministère de la Recherche, projet RNTL CLEOPATRE, grant number 01 K 0742.

1 Introduction

Un système temps-réel est un système dans lequel l'exactitude d'un calcul dépend non seulement du résultat logique (*logical correctness*) mais également de l'instant auquel le résultat est produit (*timeliness*). Le système doit ainsi être capable de réagir suffisamment rapidement pour que la réaction ait un sens. Par conséquent, une application temps-réel implique généralement des tâches auxquelles sont associées des contraintes d'échéances temporelles.

Au cours des dernières années, des applications "nouvelles" ont émergé pour lesquelles il n'est pas nécessaire de respecter toutes les échéances des tâches pourvu que les violations d'échéances soient suffisamment espacées dans le temps. Ces applications émergentes dans les domaines du multimedia et du contrôle automatique notamment, affichent des contraintes temps-réel *firm*. En d'autres termes, les violations d'échéances autorisées entraînent uniquement des dégradations de Qualité de Service (QoS) sans compromettre le bon fonctionnement du système et sans mettre en danger son intégrité. La performance du système est localement ou globalement dégradée tout en conservant ce dernier dans un état *sécuritaire*.

Considérons par exemple un système d'antiblocage des roues dans lequel typiquement, une tâche temps-réel détermine le début du freinage en scrutant périodiquement la vitesse de rotation échantillonnée de chaque roue. Dans un tel système, on se rend bien compte qu'il n'est pas nécessaire que toutes les instances de la tâche s'exécutent dans le respect de leurs échéances. Cependant, il faut veiller à limiter le nombre d'instances consécutives qui violent leurs échéances.

Les travaux concernant cette problématique ont été initiés par Hamdaoui et Ramanathan [1] par la définition d'échéances *(m,k)-firm*, selon lesquelles chaque tâche périodique doit respecter au moins m échéances sur une fenêtre de k activations. Koren et Shaha [2] introduirent également la notion de facteur de pertes (*skip factor*) pour gérer des systèmes surchargés notamment. Si une tâche possède un paramètre de pertes s_i alors, elle écartera l'exécution d'une invocation parmi s_i invocations. Il s'agit en fait d'un cas particulier du modèle *(m,k)-firm* où $m = k - 1$. Cette approche fournit une solution au problème de l'ordonnancement de systèmes surchargés, tout en représentant une garantie de QoS pour les applications temps-réel.

Caccamo et Buttazzo dans [3, 4] se sont appuyés sur ce modèle en ordonnançant des ensembles de tâches hybrides constituées de tâches périodiques tolérant des pertes et de tâches apériodiques à contraintes relatives.

La problématique décrite dans cet article est proche de cette dernière dans le sens où elle repose sur la minimisation du temps de réponse de tâches apériodiques à contraintes strictes ou relatives, en utilisant le serveur Earliest Deadline as Late as possible (EDL), sous des contraintes de pertes au sens Skip-Over. L'approche tend à distribuer le temps CPU gagné du fait des pertes, de manière à améliorer le temps de réponse des requêtes apériodiques.

2 L'ordonnancement de tâches périodiques avec QoS

2.1 Le modèle Skip-Over

Dans ce modèle à pertes sporadiques, chaque tâche périodique T_i est caractérisée par une durée d'exécution dans le pire cas c_i, une période p_i, une échéance relative égale à sa période, et un paramètre de pertes s_i, $2 \leq s_i \leq \infty$, qui fournit la tolérance de la tâche à manquer ses échéances. Selon la terminologie introduite par Koren et Shasha dans [2], chaque instance de tâche peut être *rouge* (*red*) ou *bleue* (*blue*). Une instance de tâche *rouge* doit être accomplie avant son échéance ; une instance de tâche *bleue* peut être abandonnée à tout moment.

Le modèle répond à un certain nombre de règles élémentaires que nous énonçons ci-après :

- la distance entre 2 pertes consécutives doit être d'au moins s_i périodes.
- si une instance de tâche *bleue* est abandonnée, les $s_i - 1$ prochaines instances de la tâche doivent être *rouges*.
- si une instance de tâche *bleue* s'accomplit dans le respect de son échéance, la prochaine instance de la tâche reste *bleue*.
- les $s_i - 1$ premières instances de chaque tâche doivent être *rouges*.

Lorsque $s_i = \infty$, aucune perte n'est autorisée.

2.2 Les algorithmes d'ordonnancement

2.2.1 RTO (Red Tasks Only)

Cet algorithme rejette toujours les tâches *bleues*. Les tâches *rouges* sont ordonnancées au plus tôt suivant Earliest Deadline First (la tâche ayant l'échéance la plus proche est exécutée en priorité). RTO "perd" des échéances selon un mode régulier, à savoir que la distance entre 2 pertes est exactement s_i périodes. Voici un exemple illustrant le fonctionnement de *RTO* ($c_i = 1$, $p_i = 2$, $d_i = 2$, $s_i = 3$) :

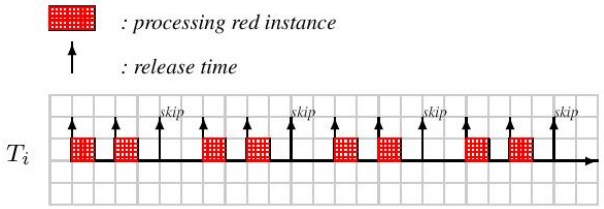

FIG 1. Exemple de pertes sporadiques sous RTO

2.2.2 BWP (Blue When Possible)

Cet autre algorithme, plus flexible, ordonnance des tâches bleues lorsque cela n'empêche pas les tâches rouges de respecter leurs échéances. L'algorithme évolue ainsi : ordonnancer les tâches rouges selon *EDF*. S'il n'y a aucune tâche rouge prête, alors activer une tâche bleue. S'il y a plus d'une tâche bleue prête alors en choisir une parmi diverses *heuristiques* [2] : la tâche bleue dont l'échéance est la plus éloignée, la tâche bleue dont l'échéance est la plus proche, ajouter une tâche au hasard et regarder s'il elle n'introduit pas de surcharge, etc. De même, illustrons l'algorithme *BWP* sur un exemple ($c_i = 1$, $p_i = 2$, $d_i = 2$, $s_i = 3$) :

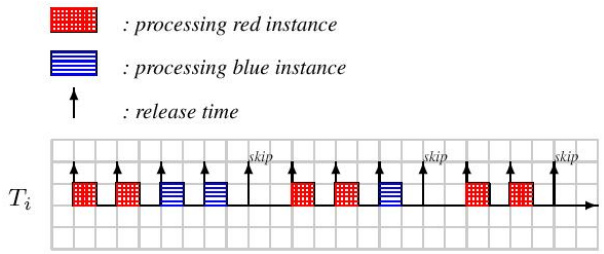

FIG 2. Exemple de pertes sporadiques sous BWP

3 L'ordonnancement de tâches apériodiques avec le serveur EDL

Le serveur Earliest Deadline as Late as possible (EDL) repose sur un algorithme d'ordonnancement dynamique, capable d'assurer la gestion de tâches apériodiques au sein d'une application temps-réel. Il consiste à exécuter les tâches périodiques au plus tôt lorsqu'il n'y a aucune activité apériodique. Dans le cas contraire, chaque fois qu'une requête apériodique survient, toutes les tâches périodiques sont ordonnancées au plus tard, dans le respect de l'ensemble des échéances des tâches. En d'autres termes, l'algorithme EDL tire profit de la laxité effective (c'est-à-dire de l'intervalle entre la date de fin d'exécution et l'échéance) des tâches périodiques, afin de minimiser le temps de réponse des apériodiques.

Dans [5], Chetto et Chetto présentent une méthode simple pour déterminer la localisation et la durée des temps creux dans n'importe quelle fenêtre d'une séquence produite par EDL. La terminologie utilisée par les auteurs est la suivante: f_Y^X représente la fonction de disponibilité définie vis à vis d'un ensemble de tâche Y et d'un algorithme d'ordonnancement X,

$$f_Y^X(t) = \begin{cases} 1 & \text{si le processeur est inactif à } t \\ 0 & \text{sinon} \end{cases}$$

Pour tous instants t_1 and t_2, l'intégrale $\int_{t_1}^{t_2} f_Y^X(t)dt$ fournit le nombre total d'unités de temps creux disponibles dans l'intervalle de temps $[t_1,t_2]$, désigné par $\Omega_Y^X(t_1,t_2)$. Il a été démontré [6] que le serveur EDL nécessite un calcul en-ligne des temps creux du processeur uniquement aux instants relatifs à l'occurrence d'une nouvelle tâche apériodique. La propriété fondamentale du serveur EDL est qu'il garantit le maximum de temps creux dans un intervalle donné pour n'importe quel ensemble de tâches. EDL a été démontré optimal [6].

THEOREME 1 *Soit X un algorithme d'ordonnancement préemptif quelconque. A tout instant t,*

$$\Omega_{\mathcal{T}}^X(0,t) \leq \Omega_{\mathcal{T}}^{EDL}(0,t)$$

Un autre résultat significatif réside dans la complexité de l'établissement de la séquence EDL. Le calcul en-ligne des temps creux [6] s'effectue en $O(\lfloor \frac{R}{p} \rfloor n)$ où n désigne le nombre de tâches périodiques, R l'échéance la plus éloignée parmi les tâches actives et p la plus petite période.

4 Utilisation de EDL pour des tâches périodiques avec QoS

4.1 EDL avec un modèle de tâches RTO

Considérons un ensemble $\mathcal{T}$ constitué de deux tâches périodiques T_1 et T_2 de couple (durée d'exécution, échéance) de (4,10) et (4,6) respectivement, et affichant un paramètre de pertes égal à 2. On suppose à présent qu'une requête apériodique de durée 5 unités de temps survient au temps $\tau = 12$. Nous avons vu précédemment que les temps creux libérés grâce aux pertes introduites au niveau des tâches périodiques, pouvaient être exploitées pour exécuter des tâches supplémentaires et plus particulièrement pour ordonnancer des requêtes apériodiques. Ces tâches apériodiques sont alors exécutées au plus tôt, tandis que les activités périodiques sont reportées plus tard dans le temps, toujours dans le respect de leurs échéances.

La séquence effectivement exécutée est présentée figure 3. Le temps de réponse observé pour la tâche apériodique est de 9 unités de temps. A partir de l'instant $t = 21$, lorsqu'il n'y a plus de tâches apériodiques au sein du système, les tâches périodiques sont de nouveau ordonnancées au plus tôt.

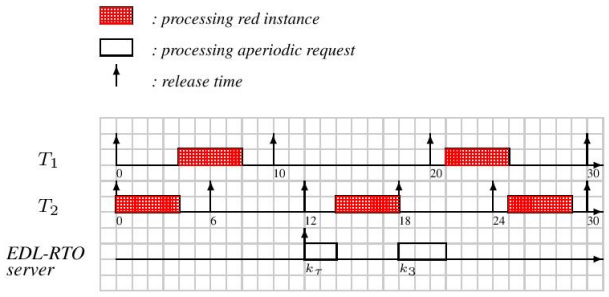

FIG 3. Séquence d'exécution d'une requête apériodique sous EDL-RTO

4.2 EDL avec un modèle de tâches BWP

Considérons à présent le même ensemble de tâches hybrides que celui utilisé précédemment pour illustrer le serveur EDL avec un modèle de tâches RTO.

L'exécution au plus tôt de la requête apériodique est illustrée sur la figure 4. Le temps de réponse pour la tâche apériodique dans ce cas est de 5 unités de temps, ce qui est largement inférieur à celui observé pour EDL-RTO. Les instances bleues de T_1 et T_2, considérées comme des temps creux et respectivement réveillées aux temps $t = 10$ et $t = 12$, sont abandonnées au terme de leurs échéances, du fait de l'exécution de la tâche apériodique. A partir de l'instant $t = 17$, lorsqu'il n'y a plus de tâches apériodiques au sein du système, les tâches périodiques sont de nouveau ordonnancées au plus tôt. On peut noter que les violations d'échéances bleues survenant aux temps $t = 18$ et $t = 20$, impliquent obligatoirement des instances de type rouge pour les $s_i - 1$ instances suivantes, de manière à respecter les contraintes de QoS au niveau des tâches.

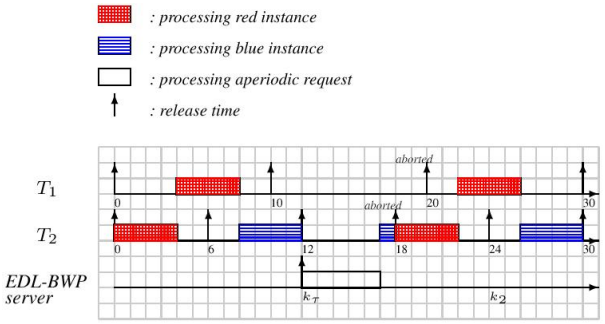

FIG 4. Séquence d'exécution d'une requête apériodique sous EDL-BWP

5 Conclusion

Les travaux présentés dans cet article proposent une solution au problème de l'ordonnancement dynamique d'ensembles de tâches hybrides définies sous des contraintes de qualité de service. Le travail effectué s'inscrit dans une perspective de gestion des cas de surcharge dans les systèmes centralisés monoprocesseur. Cette recherche vise à répondre aux besoins des nouvelles applications temps-réel dans les domaines du multimédia et du contrôle automatique, pour lesquelles l'ordonnancement temps-réel à contraintes strictes semble trop restrictif.

Dans ce contexte, nous avons présenté la fusion de deux approches existantes que sont le modèle à pertes contraintes Skip-Over et le serveur de tâches apériodiques EDL. L'objectif poursuivi consistait à fournir un modèle d'ordonnancement flexible visant à minimiser le temps de réponse des requêtes apériodiques en tirant profit des pertes autorisées au niveau des tâches périodiques. Les travaux futurs s'orientent vers l'établissement d'un nouvel algorithme à pertes, optimal du point de vue du nombre de requêtes périodiques satisfaisant leurs échéances.

Références

[1] M Hamdaoui and P Ramanathan. A dynamic priority assignment technique for streams with (m,k)-firm deadlines. *IEEE Transcations on Computers*, 44(4):1443–1451, December 1995.

[2] G Koren and D Shasha. Skip-over algorithms and complexity for overloaded systems that allow skips. *16th IEEE Real-Time Systems Symposium*, 1995.

[3] M Caccamo and G Buttazzo. Exploiting skips in periodic tasks for enhancing aperiodic responsivess. *18th IEEE Real-Time Systems Symposium*, December 1997.

[4] G Buttazzo and M Caccamo. Minimizing aperiodic response times in a firm real-time environment. *IEEE Trans. Software Eng.*, 25(1):22–32, 1999.

[5] H Chetto and M Chetto. Some results of the earliest deadline scheduling algorithm. *IEEE Transactions on Software Engineering*, 15(10):1261–1269, October 1989.

[6] M Silly. The edl server for scheduling periodic and soft aperiodic tasks with resource constraints. *The Journal of Real-Time Systems, Kluwer Academic Publishers*, 17:1–25, 1999.