Ordonnancement multiprocesseur global basé sur la laxité avec migrations restreintes

Frédéric Fauberteau, Laurent George, Damien Masson, Serge Midonnet

Ordonnancement multiprocesseur global basé sur la laxité avec migrations restreintes

Frédéric Fauberteau[1], Laurent George[2], Damien Masson[1,3], Serge Midonnet[1]

[1] Université Paris-Est / LIGM, UMR CNRS 8049
5, bd Descartes, Champs-sur-Marne, 77454 Marne-la-Vallée Cedex 2, France
{frederic.fauberteau,damien.masson,serge.midonnet@}univ-mlv.fr
[2] ECE / LACSC, 37, quai de Grenelle, 75015 Paris, France
lgeorge@ieee.org
[3] ESIEE Engineering / Département Systèmes Embarqués
Cité Descartes - BP 99, 93162 Noisy-le-Grand Cedex, France

1 Introduction

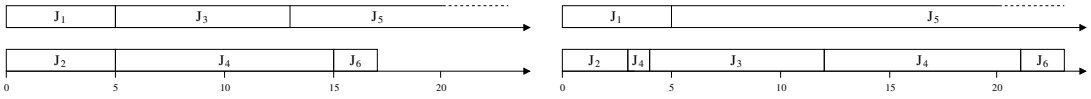
Dans cet article, nous considérons le problème de l'ordonnancement en priorités fixes de tâches temps réel en contexte multiprocesseur. Les tâches considérées sont des tâches répétitives pour lesquelles deux instances successives d'une même tâche sont activées avec une inter-arrivée minimale. En ordonnancement temps réel multiprocesseur, on distingue deux principales approches, l'ordonnancement par partitionnement et l'ordonnancement global. Dans la première approche, l'ensemble des tâches est partitionné de telle sorte que chaque sous-ensemble de tâches puisse être ordonnancé sur un processeur. Aucune migration n'est alors autorisée. Dans la seconde, les tâches peuvent migrer d'un processeur à un autre. Nous nous intéressons à cette dernière approche, mais en considérant le cas où la migration est restreinte : la migration n'est possible qu'entre les instances d'une même tâche mais pas en cours d'exécution d'une instance. Cette approche est connue dans la littérature comme ordonnancement à migrations restreintes [1]. Dans ce contexte, des anomalies sur les pires temps d'exécution peuvent se produire : un système ordonnançable pour certaines durées d'exécution peut devenir non ordonnançable avec des durées d'exécution plus petites. Nous proposons dans cet article un ordonnancement à migrations restreintes prédictible (sans anomalie) qui corrige ce problème.

2 Prédictibilité

Définition 1 (Prédictibilité de l'ordonnancement) *Un ordonnancement sera dit prédictible si pour tout jeu de tâches ordonnançable pour une configuration de pires durées d'exécution, il reste ordonnançable pour des durées d'exécution inférieures.*

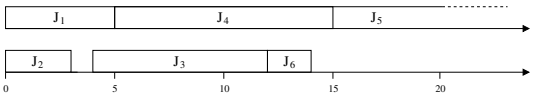
Il a été montré qu'un ordonnancement à migrations restreintes en priorités fixes n'est pas prédictible [2]. Dans la Fig. 1 est représenté le contre-exemple donné par Ha et Liu pour montrer la non prédictibilité de cette approche. Les instances J_i d'une tâche sont caractérisées par leur instant d'activation r_i, leur durée d'exécution e_i et leur échéance relative d_i. Plus l'indice i est petit, plus la priorité de la tâche est grande. Dans la Fig. 1(a), l'instance J_4 termine à l'instant 13, avant son échéance à l'instant 20. Dans la Fig. 1(b), J_4 se termine à l'instant 21 et rate donc son échéance. On constate que cela est dû au fait que J_3 est activée à l'instant 4 ; comme J_2 termine plus tôt que dans le scénario décrit dans la Fig. 1(a), J_3 peut démarrer à l'instant 4. J_3 est plus prioritaire que J_4, il préempte donc l'exécution de ce dernier. L'exécution de J_3 est retardée de 8 unités de temps, ce qui le conduit à rater son échéance.

Instance	r_i	e_i	d_i	Instance	r_i	e_i	d_i
J_1	0	5	10	J_4	0	10	20
J_2	0	{3,5}	10	J_5	5	100	200
J_3	4	8	15	J_6	7	2	25



(a) J_4 termine avant son échéance.

(b) J_4 rate son échéance.



(c) J_4 termine avant son échéance.

FIG. 1 – Ordonnancement non prédictible (a), (b) et prédictible (c).

Ce comportement s'explique par le fonctionnement de l'algorithme. Lorsque qu'une instance est active, elle est allouée sur le premier processeur disponible. En priorités fixes, un processeur est disponible s'il n'est pas en train d'exécuter une instance de priorité supérieure. Si aucun processeur n'est disponible, l'instance est mise en attente dans une file et ne pourra être démarrée que lorsqu'un processeur sera disponible. Pour pallier à ce problème, nous proposons une approche d'ordonnancement différente. Lorsqu'une instance est activée, elle est allouée à un processeur, même si elle ne peut pas démarrer son exécution à l'instant de son activation. Cette allocation est faite en considérant la laxité des instances déjà allouées aux processeurs, qu'elles soient de plus hautes ou de plus basses priorités.

Définition 2 (Laxité) *Nous définissons la laxité d'une instance comme étant la durée entre la fin de son exécution et son échéance. Lorsqu'une instance est activée, sa laxité correspond à son échéance moins sa durée d'exécution plus la somme des durées d'exécution restantes des instances de plus haute priorité.*

Ainsi, en admettant une instance sur un processeur, nous garantissions qu'aucune instance déjà admise ne ratera son échéance. De plus, pour éviter les anomalies d'ordonnancement causées lors de la réduction des durées d'exécution, nous allouons les instances sur le processeur en nous basant sur la laxité pire cas. Cette laxité pire cas est la laxité calculée en considérant le pire temps d'exécution des instances. De cette manière, les décisions qui sont prises au moment de l'allocation sont les mêmes, que les durées d'exécution soient égales ou inférieures aux pires durées d'exécution. L'application de cet algorithme conduit l'ordonnancement représenté dans la Fig. 1(c) où aucune instance ne rate son échéance.

Nous montrons la prédictibilité de cette approche par le fait que les décisions d'ordonnancement ne dépendent pas des coûts d'exécution. Nous proposons une condition nécessaire et suffisante de faisabilité qui est malheureusement de complexité exponentielle. C'est pourquoi nous proposons aussi un test de faisabilité suffisant. Nous comparons ensuite par simulation le taux de réussite de notre algorithme par rapport à des algorithmes d'ordonnancement classiques par partitionnement et globaux.

Références

[1] Sanjoy K. Baruah and John Carpenter. Multiprocessor fixed-priority scheduling with restricted interprocessor migrations. In *Proceedings of the 15th Euromicro Conference on Real-time Systems (ECRTS)*, pages 195–202, Porto, Portugal, July 2003. IEEE Computer Society.

[2] Rhan Ha and Jane W. S. Liu. Validating timing constraints in multiprocessor and distributed real-time systems. In *Proceedings of the 14th International Conference on Distributed Computing Systems (ICDCS)*, pages 162–171, Pozman, Poland, June 1994. IEEE Computer Society.